

A TRIDENT SCHOLAR PROJECT REPORT

NO. 259

**"Active Control of Rotating
Machinery Noise Through Use
of Magnetic Bearings"**



19990122 079

**UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND**

**This document has been approved for public
release and sale; its distribution is unlimited.**

REPORT DOCUMENTATION PAGE			Form Approved OMB no. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour of response, including the time for reviewing instructions, searching existing data sources, gathering and reviewing the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Avenue, Washington, DC 20540-6001.</small>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE		3. REPORT TYPE AND DATES COVERED
4. TITLE AND SUBTITLE "Active control of rotating machinery noise through use of magnetic bearings"				5. FUNDING NUMBERS
6. AUTHOR(S) John S. Wiggins				
7. PERFORMING ORGANIZATIONS NAME(S) AND ADDRESS(ES) U.S. Naval Academy, Annapolis, MD				8. PERFORMING ORGANIZATION REPORT NUMBER USNA Trident report; no. 259 (1998)
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES Accepted by the U.S. Trident Scholar Committee				
12a. DISTRIBUTION/AVAILABILITY STATEMENT This document has been approved for public release; its distribution is UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) This paper investigates magnetic bearings as an actuator for noise control. The apparatus used consists of a DC motor connected to a fan blade by a short, rigid shaft supported radially and axially by magnetic bearings. The bearings provide position control of the shaft, and thus the fan blade as well. This position control was used to vibrate the fan blade as an effective speaker for producing the secondary sound in an active sound control scheme. Due to the proximity of this secondary noise source to the primary noise sources - the motor and fan blade - good control of low-frequency noise can be achieved at all points in space. For this project, the objective was tonal noise control at frequencies corresponding to blade rate, as indicated by tachometer feedback. Noise control algorithms such as the least mean squares algorithm were implemented on a dedicated digital signal processor, with the output translated into position commands for the magnetic bearing controller. The use of magnetic bearings as an actuator for noise control is demonstrated, with tonal noise reduction of up to 6 dB achieved. This project has application to rotating machinery and marine propulsion systems.				
14. SUBJECT TERMS Active noise control, magnetic bearings, adaptive digital filtering				15. NUMBER OF PAGES 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	

U.S.N.A.—Trident Scholar project report; no. 259 (1998)

**"ACTIVE CONTROL OF ROTATING MACHINERY NOISE THROUGH USE
OF MAGNETIC BEARINGS"**

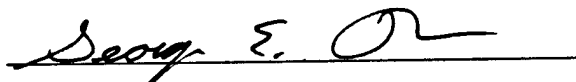
by

Midshipman John S. Wiggins, Class of 1998
United States Naval Academy
Annapolis, MD 21402



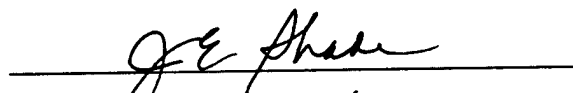
Certification of Adviser Approval

Assistant Professor George E. Piper
Department of Weapons and Systems Engineering


5/4/98

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Chair, Trident Scholar Committee


5/4/98

USNA-1531-2

ABSTRACT

This paper investigates magnetic bearings as an actuator for noise control. The apparatus used consists of a DC motor connected to a fan blade by a short, rigid shaft supported radially and axially by magnetic bearings. The bearings provide position control of the shaft, and thus the fan blade as well. This position control was used to vibrate the fan blade as an effective speaker for producing the secondary sound in an active sound control scheme. Due to the proximity of this secondary noise source to the primary noise sources—the motor and fan blade—good control of low-frequency noise can be achieved at all points in space. For this project, the objective was tonal noise control at frequencies corresponding to blade rate, as indicated by tachometer feedback. Noise control algorithms such as the least mean squares algorithm were implemented on a dedicated digital signal processor, with the output translated into position commands for the magnetic bearing controller. The use of magnetic bearings as an actuator for noise control is demonstrated, with tonal noise reduction of up to 6 dB achieved. This project has application to rotating machinery and marine propulsion systems.

KEY WORDS

Active Noise Control, Magnetic Bearings, Adaptive Digital Filtering

NOMENCLATURE

Actuator: In a control system, the element that converts the output of the controller into a useful signal. In a noise control system, the actuator converts the controller output—a voltage—into a sound.

Air Gap: In a magnetic bearing, the distance between the bearing and the rotor.

Analog: A system with a continuous time, continuous value output.

Band-pass Filter: A filter that only allows inputs within a certain frequency range to pass.

Bearing: A device that supports the rotating components of a rotating mechanical system.

Blade Rate: The frequency at which the blades of a fan or propeller pass any point along their path. The blade rate is the product of the shaft rate and the number of blades.

Bode Plot: A plot of the frequency response of a system. The gain and phase are plotted separately, with the gain measured in decibels, and the phase measured in degrees.

Broadband: Consisting of a large range of frequencies.

C: A computer programming language.

Closed Loop System: A system that uses feedback.

Continuous-Time: A system whose output changes continually. Most analog circuits, such as a controller built from op amps, are continuous-time systems. Continuous-time systems are analyzed using the Laplace transform.

Controller: A device, usually electronic, that regulates the behavior of a system, driving the output of that system to a desired value.

Digital Signal Processor (DSP): A computer processor designed specifically for signal processing. Controller transfer functions can be implemented as a computer program run on a DSP, rather than a circuit involving op amps.

Digital: A system whose output changes between discrete values at discrete time intervals.

Discrete-Time: A system or signal whose value changes only at certain intervals. For example, the output of a DSP changes only at the sample intervals. Discrete-time systems are normally composed of digital components such as a computer or DSP, and are analyzed using the z-transform.

Disturbance Sound: In a sound control system, any noise other than the primary and secondary sounds, such as background noise.

Floating Point: A numeric representation, similar to scientific notation, used in computers and DSPs. Floating point numbers provide greater precision than other numeric representations, but take longer to manipulate in many processors.

Frequency Response: The difference in magnitude and phase between the input and output of a system, as the input is varied over a range of frequencies.

Gain Margin: The amount of gain that can be added to a system before the system gain reaches 0 dB at the frequency at which the phase is 180° .

Laplace Transform: A mathematical transform that converts linear differential equations into algebraic polynomials.

Linear: A system whose output is directly proportional to parameters such as velocity, force, position, and current.

Narrowband: Consisting of a small range of frequencies.

Nonlinear: A system whose output is not directly proportional to system parameters. Instead, the system output may be related to the square or inverse of some parameters.

Open Loop System: A system that does not use feedback.

Operational Amplifier (Op Amp): A fundamental electronic component of control circuits. Op amps allow transfer functions to be implemented easily as electrical circuits.

Pass Band: The frequency range that a band-pass filter will pass.

Phase Margin: A measure of the robustness of a system. The amount of phase that can be added to or subtracted from a system before the system phase is 180° at the frequency at which the system gain crosses the 0 dB line.

Plant: In a control system, the plant consists of all of the existing components whose behavior the controller must regulate.

Pole: A root of the denominator of a system's transfer function. The poles of a system describe the behavior of that system.

Power Spectrum: A plot of the power in a signal broken down by frequency. Signals contain components at many frequencies, and a power spectrum gives an indication of the relative amplitudes of those frequency components.

Primary Sound: In a sound control system, the noise whose amplitude is to be reduced.

- Relative Permeability:** An indicator of the magnetic "conductivity" of a material. Materials with a high relative permeability are capable of sustaining stronger magnetic fields.
- Robustness:** Ability of a system to maintain its desired behavior in the presence of disturbances, variations, and uncertainties.
- Root Locus:** A plot of the various possible poles of a system as a function of some system parameter, usually the controller gain. As the gain of a closed-loop system increases, the pole locations move from the locations of the open-loop poles to the locations of the open-loop zeros.
- Sampling Time:** The length of time between samples of a digital system. At each sample, the input value of the system is measured, and a new output value is asserted.
- Saturation:** A magnetically saturated material cannot support any increase in its current magnetic field. An electronically saturated component has reached its peak output and cannot output any higher voltage or current.
- Secondary Path:** In an active noise control system, all of the transfer functions from the reference signal input through the error microphone.
- Secondary Sound:** The sound produced by a noise control system to attenuate the primary sound.
- Shaft Rate:** The frequency of rotation of a shaft. A shaft that rotates ten times each second has a 10 Hz shaft rate.
- Tachometer:** A device that senses rotational speed.
- Tap:** A single weight in a weight vector.
- Tone:** A pure tone is a sound or signal containing only one frequency.
- Transfer Function:** An equation that describes the behavior of a system or component. A transfer function is the relationship between the input to a system and the corresponding output, expressed as a ratio of differential equations using the Laplace transform.
- Weight Vector:** The set of weights of a finite impulse response filter. The value of the weights determines the characteristics of the filter.
- z-Transform:** A mathematical transform related to the Laplace transform, but designed specifically for use in analyzing discrete-time systems.
- Zero:** A root of the denominator of a system's transfer function.

TABLE OF CONTENTS

ABSTRACT	I
KEY WORDS	I
NOMENCLATURE	II
1 BACKGROUND	1
2 MAGNETIC BEARINGS	5
2.1 THE ELECTROMAGNETIC PLANT	5
2.2 CONTROLLER DESIGN	10
2.2.A MODELING THE PLANT	10
2.2.B CONTROLLER SELECTION	14
2.2.C ROOT LOCUS DESIGN METHODS	16
2.2.D FREQUENCY RESPONSE DESIGN METHODS	24
2.3 IMPLEMENTATION	27
2.3.A CONTROLLER	27
2.3.B AMPLIFIERS	27
2.3.C SENSORS	28
2.4 SIMULATION	29
2.5 BEHAVIOR OF THE SYSTEM	31
3 ACTIVE SOUND CONTROL	32
3.1 ADAPTIVE DIGITAL FILTERING	33
3.1.A LEAST MEAN SQUARES	36
3.1.B FILTERED-X	38
3.2 SIMULATION OF THE NOISE CONTROL SYSTEM	41
3.3 IMPLEMENTATION	45
3.3.A DIGITAL SIGNAL PROCESSORS	45
3.3.B GENERATION OF A FEEDFORWARD SIGNAL	45
3.3.C CONTROL ALGORITHM	46
3.3.D ERROR SIGNAL FEEDBACK	47
4 ACOUSTICS	48
4.1 PRIMARY NOISE SOURCES	48
4.2 SECONDARY NOISE SOURCE	48

4.3	MECHANISM OF SOUND CONTROL	49
5	RESULTS	53
5.1	ACTUATOR TESTS	54
5.2	NOISE CONTROL SYSTEM TESTS	55
5.3	GLOBAL EFFECTS TESTS	60
6	SUMMARY	61
APPENDIX A—LINEAR APPROXIMATION PROGRAM		62
APPENDIX B—CONTROLLER DESIGN PROGRAM		65
APPENDIX C—MAGNETIC BEARING CONTROLLER CIRCUIT		68
APPENDIX D—AMPLIFIER SPECIFICATIONS		71
APPENDIX E—MAGNETIC BEARING SIMULATION		72
APPENDIX F—NOISE CONTROL SIMULATION		88
APPENDIX G—SHARC PROCESSOR SPECIFICATIONS		106
APPENDIX H—REFERENCE SIGNAL GENERATION PROGRAM		108
APPENDIX I—FILTERED-X PROGRAM		120
REFERENCES		132

1 BACKGROUND

It is often desirable to control the noise emissions of rotating machinery, whether in industry, either for safety or aesthetic purposes, or in military applications, for stealth. For example, a great deal of information about a submarine can be obtained by observing which particular frequencies of sound the submarine is producing. In order to reduce the chances of being detected, or of being accurately classified, a submarine seeks to minimize emissions of these tones. Passive dampening materials and barriers are effective and practical for reducing high frequency sound, but the bulk and weight necessary in such structures to block low frequency noise makes them impractical in many situations. Active sound control provides a viable alternative for reducing low frequency noise.

In its simplest form, active sound control eliminates a pure tone—or sine wave—from one source by producing another sine wave, of identical magnitude but opposite phase, from a second source. The original sound wave is called the “primary sound,” while the sound wave used to cancel the primary sound is the “secondary sound”. When these two waves combine, they cancel one another out as shown in figure 1. For this project, a simple electric motor and fan blade were used as the primary noise source. The normal operation of the motor and the movement of the propeller through the air generate the primary noise, which contains many frequencies. Certain frequencies, such as the rotation rate of propeller blades through the water, are particularly loud. The aim of this project was to reduce the noise level of such narrow-band tones. The secondary sound used for active noise control was produced by vibrating the shaft, which, in turn, vibrated the propeller. The propeller acted as a speaker, and transmitted its vibrations through the air as pressure waves, or sound. The mechanism for vibrating the shaft lies in the magnetic bearings that support the shaft. This technique is based on a similar actuator for sound control, which was proposed and evaluated in [18].

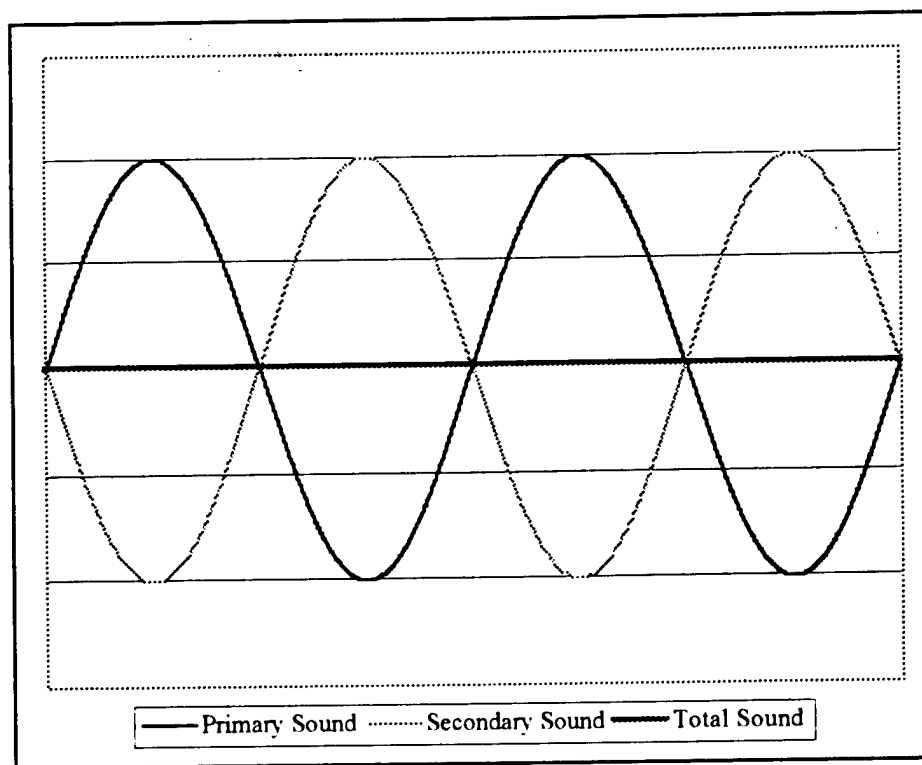


Figure 1 Cancellation of two sine waves of opposite phase

The shaft of the model propulsion system used in this project is supported by three sets of bearings—one radial bearing at each end, and a thrust bearing in the middle. Each of the radial bearings has four channels, or individual electromagnets. Two opposing channels control the vertical position of the shaft, and two control the horizontal position of the shaft. The thrust bearing controls the position of the shaft along its third axis, which is the direction of the thrust from the fan blade. Figure 2 shows a simplified schematic of the experimental apparatus, indicating the location of the magnetic bearings, and figure 3 shown the location of the magnetic bearings on the actual motor and propeller assembly. Magnetic bearings are groups of electromagnets that pull on the shaft—or rotors attached to the shaft—in order to maintain a desired shaft position. Magnetic bearings provide several advantages over conventional bearings: since the shaft is not in contact with the bearings, there is little or no friction at the bearings; the bearings do not need to be lubricated; they require no regular maintenance; and, of special significance in this project, magnetic bearings can adjust the position of the shaft which they support [22]. The force of the bearings is a function of both the current through the windings and the distance between the magnets and the rotor [22]. By controlling the current through the windings, the shaft can be held at a given position. By rapidly altering the force of the bearings, the control system can cause the shaft to vibrate with a particular frequency and magnitude. A control system has been designed

to perform this function using a digital signal processing card and an analog controller built from standard electrical components.

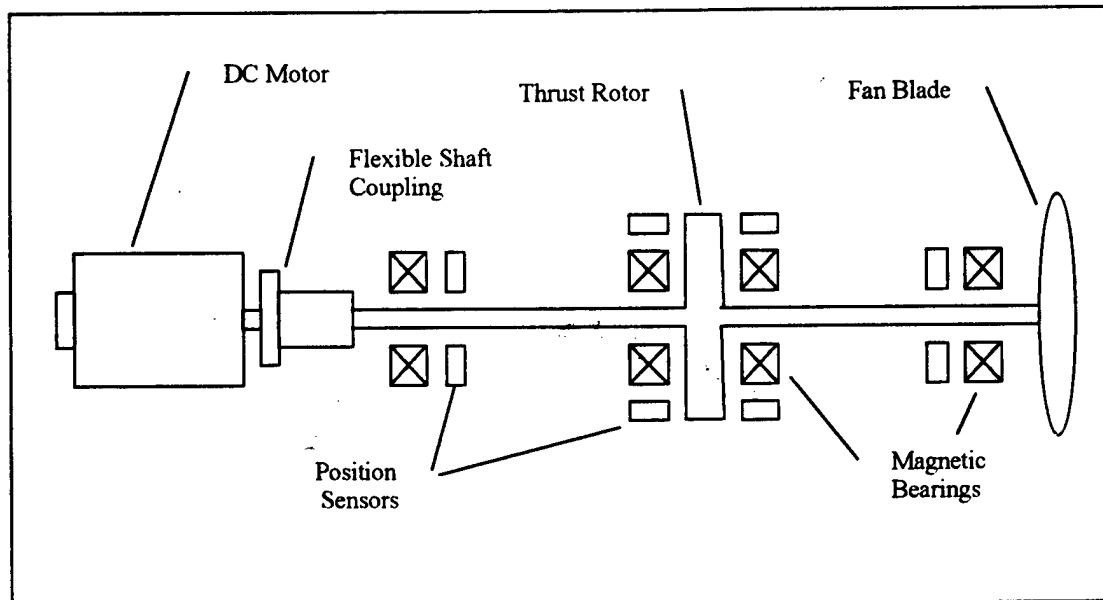


Figure 2 Schematic diagram of the experimental apparatus

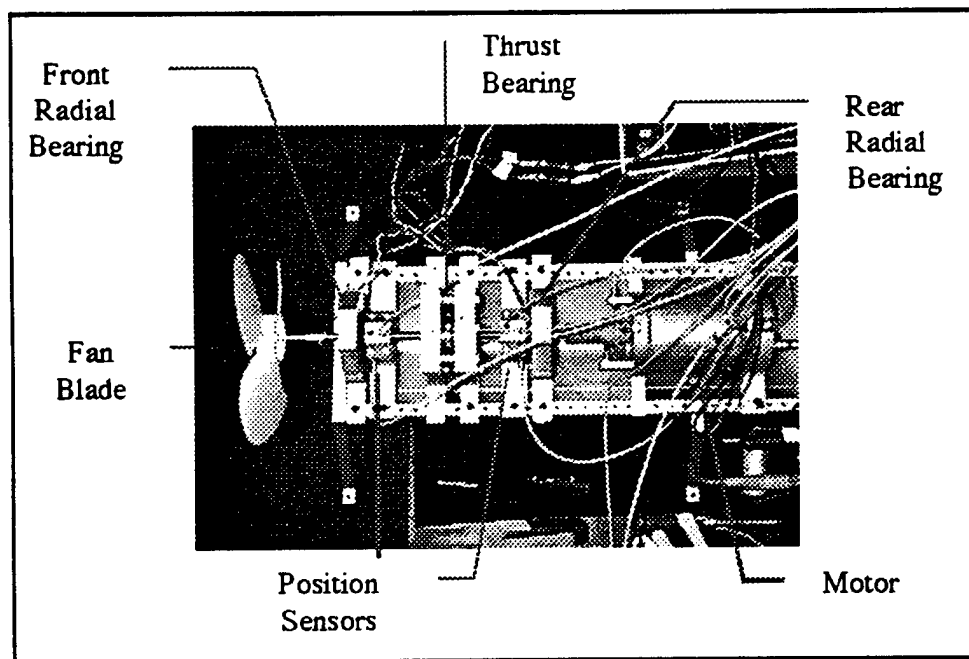


Figure 3 Photograph of the experimental apparatus, indicating locations of major components

Though not a very old technology, active sound control has been well-researched. The commonly known algorithms are fast, effective, and robust [10]. This project has not focused on developing new sound control algorithms—several common algorithms were adapted for use in this particular system. Rather, the focus of this project is the use of a new actuator—magnetic bearings—in the creation of the secondary sound, and evaluation of the effectiveness of that actuator in achieving global noise control for a rotating mechanical system.

2 MAGNETIC BEARINGS

The initial phase of this project focused on the construction of the experimental apparatus, particularly the magnetic bearings. The actual apparatus used in this project is shown in figure 4. The apparatus consists of a DC motor connected to a fan blade by a short, rigid, shaft, supported entirely by magnetic bearings. One thrust bearing supports the shaft laterally, while two radial bearings support the shaft vertically and horizontally. Each magnetic bearing consists of a group of electromagnets, each of which generates a magnetic field pulling the rotor towards it. Sensors around the shaft measure the position of the shaft, and send that information to the controller. The controller adjusts the amount of voltage or current to each electromagnet to keep the shaft in the desired position. Figure 5 depicts a simple active magnetic bearing system [22].

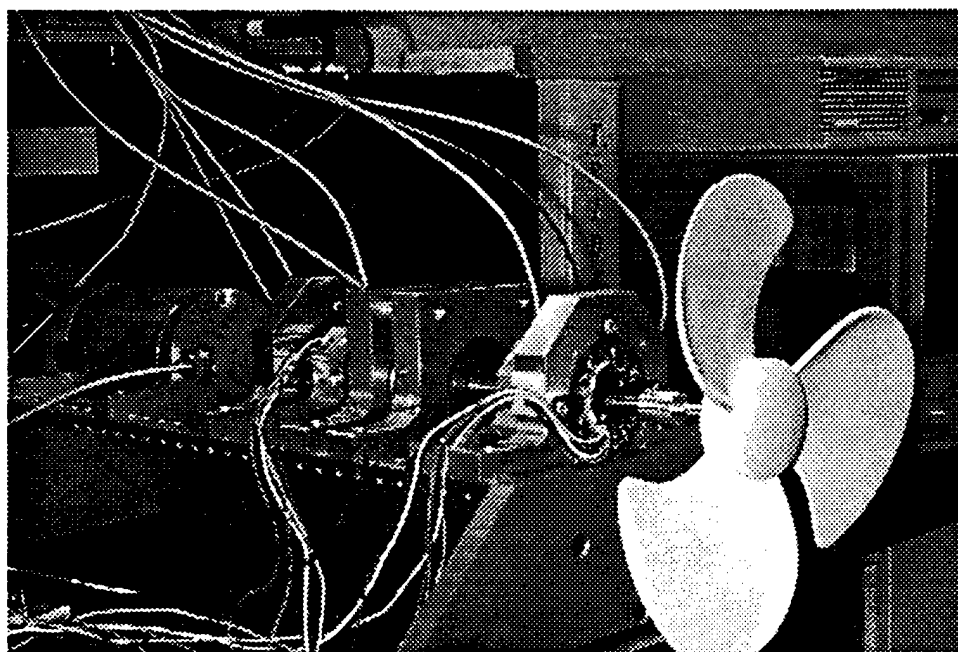


Figure 4 Experimental apparatus

2.1 THE ELECTROMAGNETIC PLANT

The theory and operation of a magnetic bearing is best demonstrated by examining the design process for a bearing, such as the thrust bearing for the experimental apparatus. There were three design criteria for the bearing: force produced by the bearing, current allowed in the windings, and dimensions. For maximum force, the bearings were designed to be as large as possible within the size constraints of the apparatus that would be supporting them. The value for current was determined by

consideration of the power amplifiers, rather than the bearing windings, since the bearings are capable of handling as much current as the amplifiers can supply. The number of windings then remained as the primary design variable by which to arrive at a sufficient force output. By plotting a three dimensional graph of the force-current-winding relationship for the ideal dimensions of the bearings (figure 6), an appropriate number of windings could be selected to meet the force requirements. With the physical limit on the number of windings due to the size of the bearing core in excess of one hundred turns, operating point force output proved to be a cheap commodity. Thus, rather than make any detailed measurements of the required force to counteract the thrust of a fan blade, a reasonable force value was chosen.

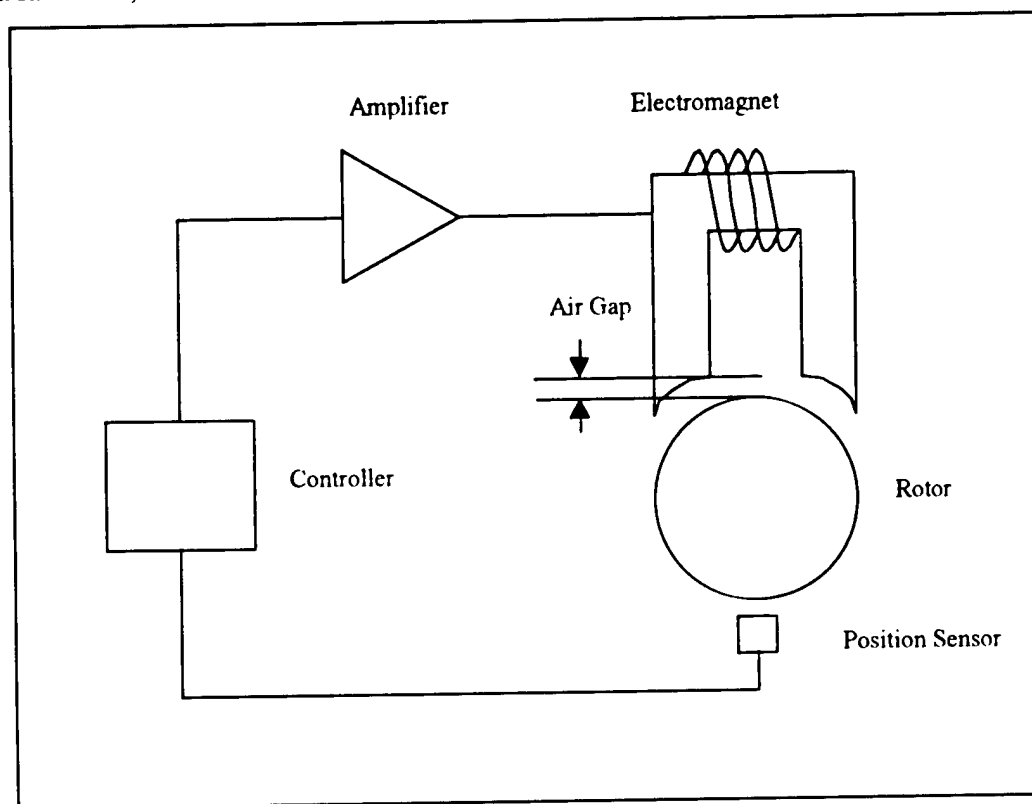


Figure 5 Simple active magnetic bearing system

The force of the bearing on the rotor is primarily a function of current through the bearing windings and distance between the bearing and the rotor. The formula for this relationship is derived here by analysis of a magnetic circuit, with certain physical relationships assumed without proof. In a magnetic bearing, the magnetic circuit is a path through the bearing core (the electromagnet in figure 5), the air gap, and the rotor. The energy stored in any portion of the magnetic circuit is the product of the magnetic

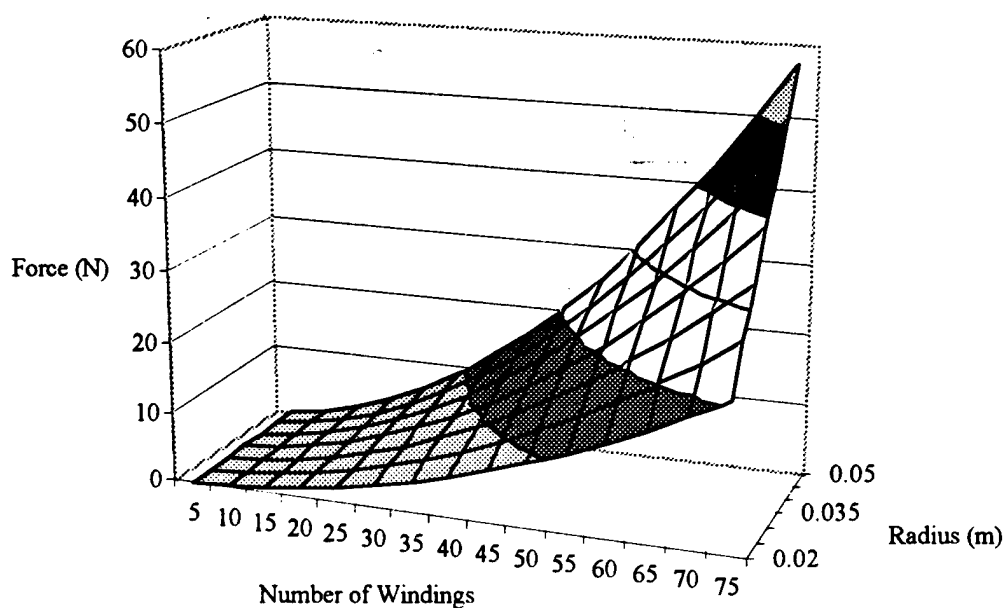


Figure 6 Thrust bearing force as a function of bearing radius and number of windings

field, H , across that portion, the flux, Φ , through that portion, and the volume of the portion. For the air gaps in particular, the equation for stored energy is¹:

$$E = B \cdot H \cdot A \cdot d \quad (2.1)$$

E : Energy

B : Flux density, Φ / m^2

H : Magnetic field

A : Cross sectional area of the magnetic circuit

d : Air gap

A change in the size of the air gap causes a loss or gain of energy in the air gap, proportional to the change in volume of that portion of the circuit. The energy added to or dissipated by the system appears in the form of a force on the bearing pieces:

$$F = \frac{\partial E}{\partial d} = B \cdot H \cdot A \quad (2.2)$$

The flux density in the bearings also changes, since it is a function of current and air gap. The relationship between flux density and current is derived from the magnetic field.

¹ This derivation is largely from [Shiau] and [Schweitzer]

The summation of the magnetic field over the entire length of the magnetic circuit is equal to the product of the current through the coil windings and the number of windings:

$$\oint H dx = \ell_{Fe} \cdot H_{Fe} + 2 \cdot d \cdot H_a = n \cdot i \quad (2.3)$$

ℓ_{Fe} : Length of the magnetic circuit through the bearing

H_{Fe} : Magnetic field in the bearing

d : Air gap portion of the magnetic circuit

H_a : Magnetic field in the air gap

The magnetic field is assumed to be the same in the air gap and the bearing pieces. This assumption is accurate for small air gaps, such as those in magnetic bearings. Flux density is related to the magnetic field by the following equation:

$$B = \mu_0 \cdot \mu_r \cdot H \quad (2.4)$$

μ_0 : Permeability of free space

μ_r : Relative permeability. $\mu_r \sim 1$ for air, $\mu_r \sim 1000$ for steel

This equality is substituted into equation (2.3) to yield the relationship between flux density, current, and air gap:

$$B = \frac{\mu_0 \cdot n \cdot i}{\ell_{Fe} / \mu_r + 2 \cdot d} \quad (2.5)$$

Substituting this equation back into the original force relationship (equation (2.2)), and using the relationship between B and H, yields force as a function of current and displacement:

$$F = \mu_0 \cdot \left(\frac{n \cdot i}{\ell_{Fe} / \mu_r + 2 \cdot d} \right)^2 \cdot A \quad (2.6)$$

For the radial bearings, an additional cosine factor is added to account for the orientation of the bearings.

The force equation used in designing the thrust bearing is the non-linear equation given above in the derivation of a plant model. The assumptions for this equation are as follows: no magnetic flux flows outside of the bearing core, the rotor, and the air gap between the two; the cross-sectional area of the bearing is constant throughout the circuit; the relative permeability of the material is constant; and the material has not reached magnetic saturation. The first assumption is somewhat inaccurate, but only results in a calculated force value that is slightly higher than the actual value. This error in the force calculation is more than made up for in the overestimate of the necessary force from the bearing. The second assumption requires that the bearing be designed with an inner section that is thicker than the outer section, so that the cross section of each is the same,

despite the different diameters. The cross sectional area is still not the same at all points in the circuit, so the cross section used in calculation is the minimum cross sectional area in the core. The relative permeability of the bearing material was assumed to be 750. Design of a high-quality bearing would call for the use of a metal with special magnetic characteristics. The two most important magnetic characteristics are relative permeability and magnetic saturation value. Iron has a high relative permeability, as is desired, but has a low saturation value—it is able to produce a strong magnetic field with little current, but quickly reaches saturation value, beyond which even large currents produce little increase in field strength. Specially designed metals, such as vanadium permendur, have both high relative permeability and high saturation value, and are excellent materials from a magnetic standpoint. However, these metals are expensive, difficult to machine, and require heat treatment. In light of these factors, and the calculations that indicated a sufficient force value from the thrust bearing would be easy to achieve, stock steel was used in construction of the thrust bearing pieces. Experimentation indicates that the steel rotor and core provide sufficient force to drive the shaft to the desired position.

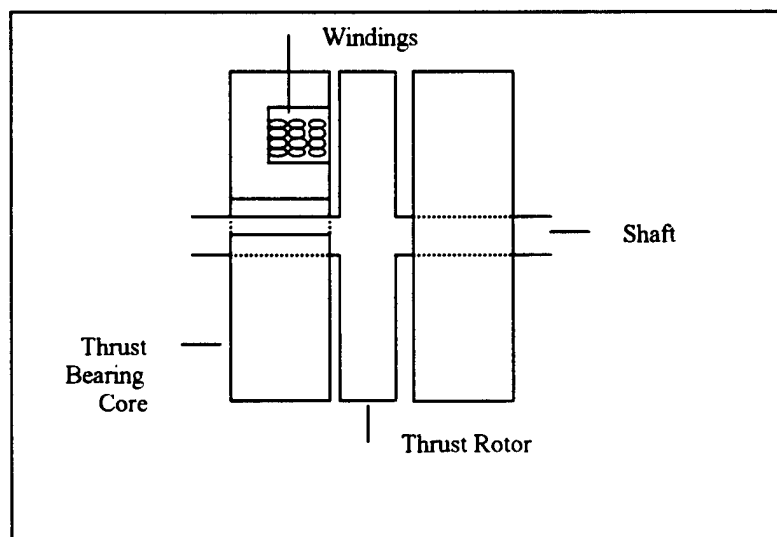


Figure 7 Thrust bearing schematic

As constructed, the thrust bearing consists of two doughnut-like pieces—the cores—with the shaft running through the center of each. Each core is slotted, as shown in figure 7. The windings, which are thinly insulated copper wire, are fitted completely within the slots. The two cores face one another on opposite sides of a flywheel, or rotor, which is affixed to the shaft. The bearings pull on the rotor in opposite directions to control the lateral position of the shaft.

2.2 CONTROLLER DESIGN

The most difficult portion of this project is the design of a controller for the magnetic bearings. Since the bearings are attractive electromagnets, the system is inherently unstable [22]. If the rotor is disturbed from equilibrium by even an infinitesimal amount, the balance of forces will be lost, and the rotor will crash into the bearing. In order to maintain an equilibrium, some mechanism of control must be provided, so that the force of the bearing returns the rotor to its desired position, rather than pulling it away.

2.2.A Modeling the Plant

The first step in controller design is to form a mathematical model of the system to be controlled. This model starts with Newton's second law of motion. The objective of the control system is to stabilize the position of the rotor. It is first assumed that the only force on the rotor is from the magnetic bearings. Thus, the acceleration of the rotor, or the second derivative of its position, is equal to the force of the bearings divided by the mass of the rotor:

$$x'' = \frac{F}{m} \quad (2.7)$$

x'' : Second derivative of position

F : Force of the bearings

m : Mass of the rotor

Equation (2.6) gives the force of the bearings as a nonlinear function of current and rotor position. However, the vast majority of existing control theory revolves around control of linear systems. Therefore, in order to develop a controller for the magnetic bearings, a linear approximation of the system must be developed. The primary assumption in linearizing the bearings, or "plant", is that for small displacements, the relationship between force, position, and current is sufficiently linear to apply existing control laws. The force is still a function of current and distance, but now it is a linear function. The linear approximation of the force equation can be written as:

$$F = k_i \cdot i + k_d \cdot d \quad (2.8)$$

Where k_i is the force-current constant and k_d is the force-displacement constant. Each element of the linear force equation is, itself, a linear estimate of the actual relationship between force and current or displacement. In order to be as accurate as possible, the linear function should have the same derivative as the non-linear function at the operating point, as shown in figure 8. There are several ways of finding the slope of the curves at the operating point. In order to have a fast and adaptable method, a computer program

was used to calculate the values. A listing of the program, which is actually a Matlab script file, is included as Appendix A. The program plots a force-current curve, holding displacement constant at its operating-point value, and a force-displacement curve, holding current constant at its operating-point value. The slope of each curve at the operating point can be found with arbitrary accuracy by finding the equation of a line through the two points on the curve immediately to either side of the operating point.

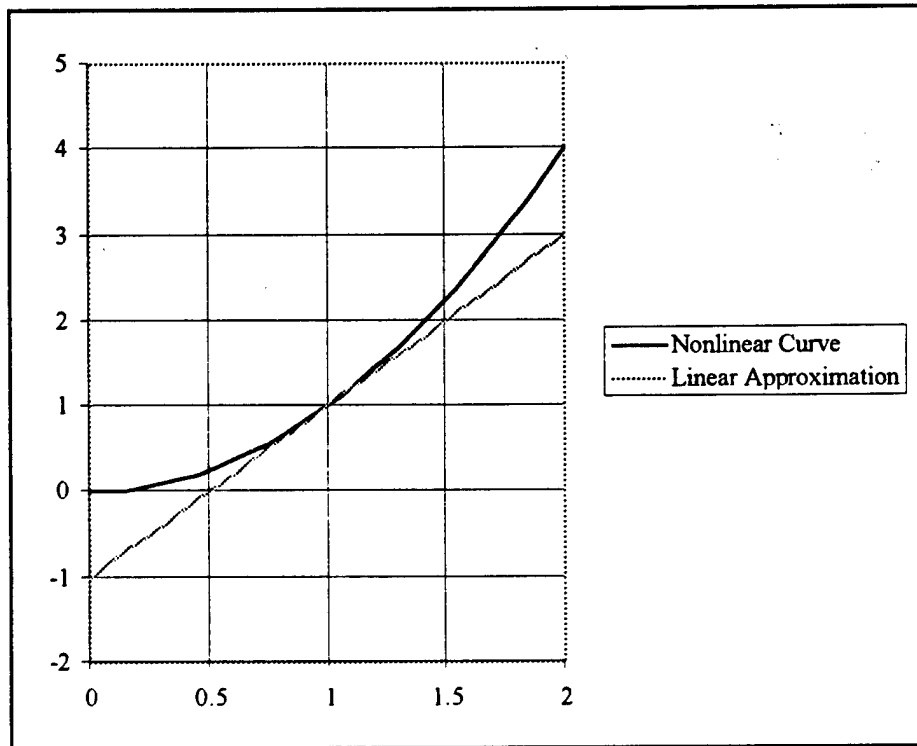


Figure 8 Linear approximation of a nonlinear function. The approximation is accurate for values near the point at which it is tangent to the actual curve.

The force of a magnetic bearing as derived above is a function of bearing current and rotor displacement. The rotor displacement is available as a variable in the original differential equation derived immediately from Newton's second law. However, the bearing current is not generally available, as the output of the driving mechanism for the bearings—that is, a control circuit or amplifier—is in the form of a voltage rather than a current. The conversion between voltage and current is assumed to be a function only of the electrical circuit formed by the bearing coils, which is a first-order R-L network²:

$$V_b = V_c - R \cdot i \quad (2.9)$$

$$V_b = L \cdot i' \quad (2.10)$$

² In reality, there is an additional voltage input caused by the movement of the rotor, which is a conductor, within the magnetic field of the bearings. The effects of this additional voltage source were examined in a nonlinear computer simulation, which is discussed later.

$$\therefore i' = \frac{V_c - R \cdot i}{L} \quad (2.11)$$

V_b : Voltage across the bearing coils

V_c : Voltage output of the controller

R : Resistance of the coils

L : Inductance of the coils

i : Current

Combination of the above differential equations is made simpler by use of the Laplace transform, which transforms time-domain differential equations to frequency-domain algebraic functions of a complex variable, "s". Multiplication by s in the frequency domain represents a derivative in the time domain. The three pertinent differential equations, after transformation, are listed below:

$$\text{Newton's second law:} \quad s^2 \cdot X(s) = \frac{F(s)}{m} \quad (2.12)$$

$$\text{Force equation:} \quad F(s) = k_i \cdot I(s) + k_d \cdot X(s) \quad (2.13)$$

$$\text{Current-voltage relationship:} \quad s \cdot I(s) = \frac{V_c(s) - R \cdot I(s)}{L} \quad (2.14)$$

$$\text{or:} \quad I(s) = \frac{V_c(s)}{L \cdot s + R} \quad (2.15)$$

Combining these to get position as a function of voltage and position yields:

$$s^2 \cdot X(s) = \frac{\frac{k_i \cdot V_c(s)}{L \cdot s + R} + k_d \cdot X(s)}{m} \quad (2.16)$$

The transfer function, or ratio of the output, $X(s)$, to the input, $V_c(s)$, is:

$$\frac{X(s)}{V_c(s)} = \frac{k_i / m}{(L \cdot s + R)(s^2 - k_d / m)} \quad (2.17)$$

This completes the model of the bearing plant transfer function from a voltage input to a position output. The linearized plant block diagram is shown in figure 9, with some modifications made as described in the following paragraphs.

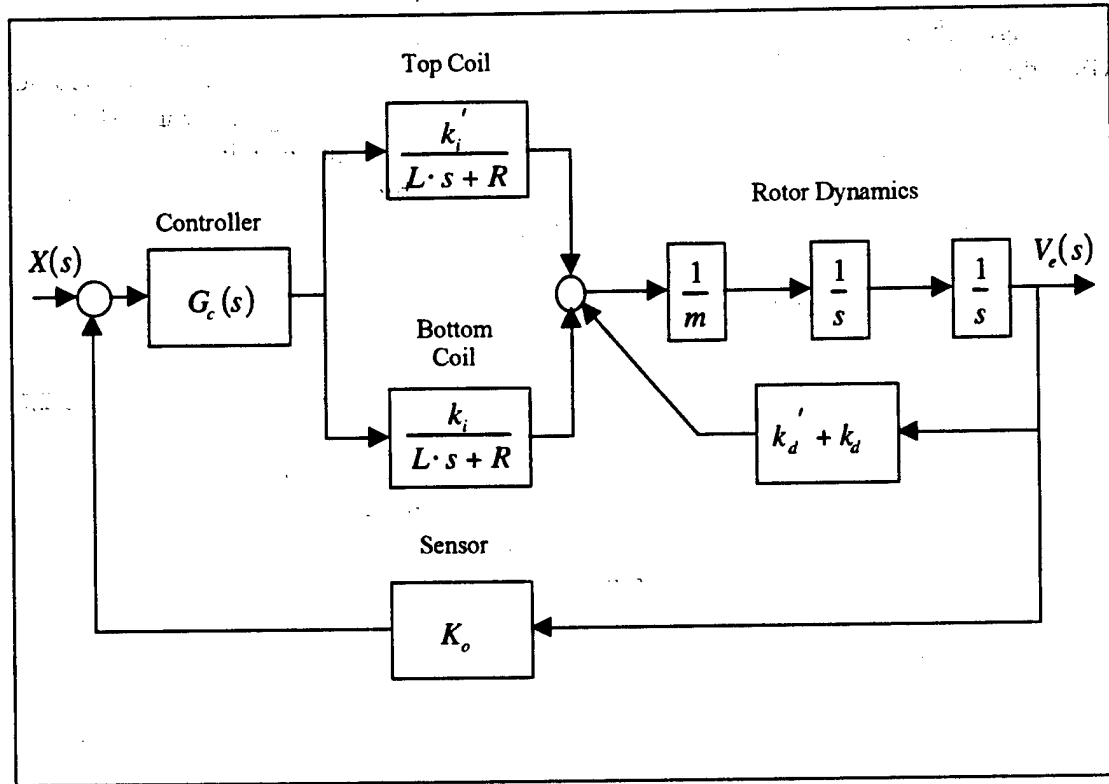


Figure 9 Linearized magnetic bearing system block diagram

The magnetic bearings are described by a third-order linear time-invariant system. The three poles, or eigenvalues, of the system are as follows:

$$s = -\frac{R}{L}$$

$$s = \pm \sqrt{\frac{k_d}{m}}$$

A pole in the s-domain represents an exponential function in the time domain. The system has one pole, or root of the denominator, with a positive real part. Poles with positive real parts indicate exponentials that will grow to infinity over time—clearly unstable system behavior. As a method of determining system behavior, it is useful to plot system poles on a Cartesian plane, with the x-axis indicating values of the real part of s, and the y-axis indicating values of the imaginary part. Any pole in the right half of the s-plane has a positive real part and thus indicates an unstable system. A plot of the poles of the uncontrolled system is shown in figure 10, and described in more detail below. With the plant modeled, a controller for the system can be selected.

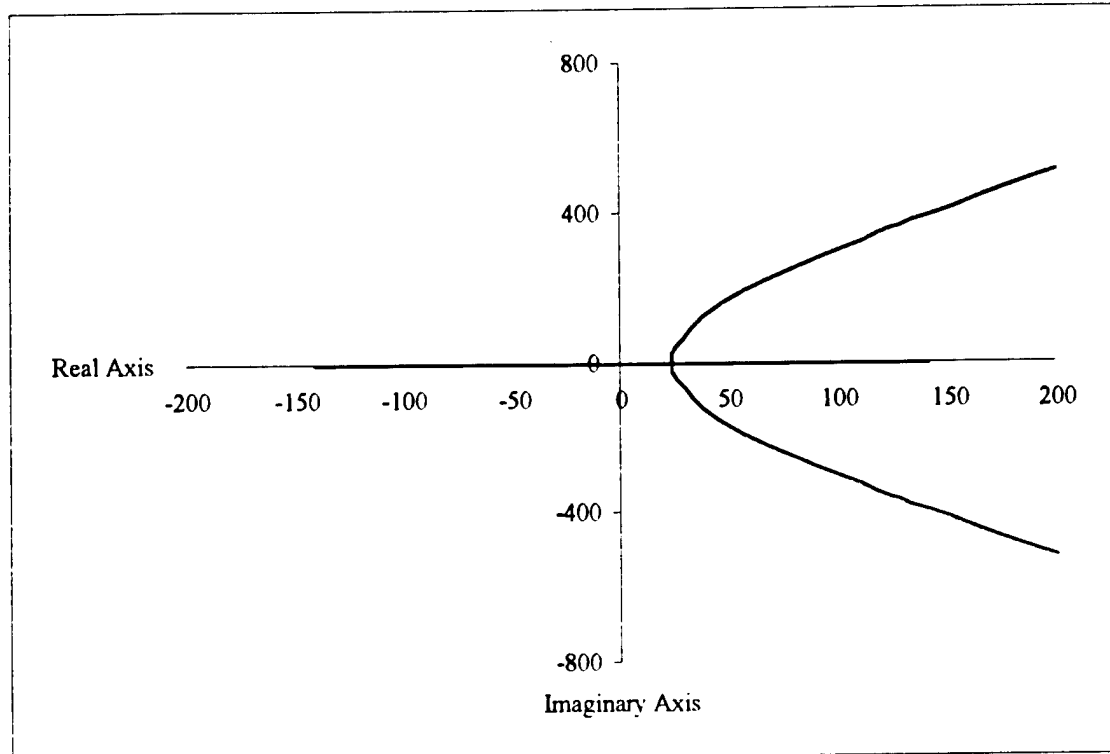


Figure 10 Root locus of uncontrolled magnetic bearing plant

2.2.B Controller Selection

Before selecting a controller, it is important to specify the design criteria. Design criteria characterize the desired behavior of the system, often in response to various control inputs. Since the magnetic bearings system is designed to maintain a constant position of the shaft, rather than track an input, design criteria such as percent overshoot, settling time, and rise time are of little importance³. Rather, the primary concern is the robustness of the design. The linear model of the bearing plant is only accurate close to the operating point, and contains a large degree of uncertainty as well as potentially significant approximation errors. Therefore, the controller must be effective in its regulation in the presence of significant parameter variations and disturbance inputs. The robustness of the controller is evaluated with two primary methods: phase margin from system frequency response, and behavior in nonlinear simulation.

Each channel of a magnetic bearing—the top, bottom, left, and right electromagnets on a radial bearing for example—must be controlled in order for the

³ The properties listed describe the response of a system to a change in input from one constant value to another. Rise time is the length of time that elapses before the system first reaches the new desired output value; percent overshoot describes how much the system output exceeds the desired output during transition, before settling on the new value; settling time is the length of time required for the system to be stabilized within two percent of the desired value.

bearing to remain stable. However, it is not necessary to have a separate controller for each channel, if a method known as "differential driving" is used. Differential driving uses one controller for each direction on each bearing. Two controllers are required for each radial bearing—one for vertical control and one for horizontal—and one controller is necessary for the thrust bearing. The use of a differential driving mode incurs several advantages: the number of controllers that must be built is reduced by half; the complexity of feedback electronics is reduced; and the force-current relationship becomes linear [1]. In a differential driving mode, the current through one side is controlled directly by the control circuitry, while the opposite side uses the negative of the controller output. Since there cannot be negative current through the bearings, a "bias current" of identical magnitude is applied to both bearings in an opposing pair, causing them both to tug at the rotor, and increasing the stiffness of the system. Any controller output will increase the current through one side of the bearing above the level of the bias current, while lowering the amount of current through the other side of the bearing. The operating-point current is different for each of the two channels in the vertical direction, since the top channel must provide a constant force to counteract the force of gravity. The force-position and force-current constants for the linear estimate are therefore different for each of the vertical channels, as is reflected in the block diagram of figure 9 with the use of k_d' and k_i' for one channel and k_d and k_i for the other.

The task now turns to the design of a controller for the bearings. In order to achieve robust control with no steady-state error in position, a controller was selected with gains proportional to the error between the current position and desired position; proportional to the derivative of that error, or the velocity of the rotor; and proportional to the integral of the error. This is known as a PID (Proportional/Integral/Derivative) controller, and is commonly used in industry for its robust characteristics [3]. The controller is described by the following differential equation:

$$v'' + \frac{v'}{\tau} = k_D \cdot e'' + k_P \cdot e' + k_I \cdot e \quad (2.18)$$

v : Controller output voltage

e : Input error signal

k_P, k_I, k_D, τ : Controller gain constants

The proportional factor is a basic gain controller, and acts like a spring. The further the rotor is from the bearing, the more force the bearing exerts, while the bearing exerts less force if the rotor is too close. A direct proportional controller cannot stabilize the magnetic bearing system. The gain will either be too small to lift the bearing, or too large, which will cause the rotor to bang from one side to the other. The latter case is an "undamped" system. In order to add damping, the derivative controller is introduced. This portion of the controller adds a voltage proportional to the velocity of the rotor. If

the rotor is moving to the left, the derivative controller exerts a force to the right. The proportional and derivative controller together could be effective. However, with a PD controller, any change in the external force on the rotor would cause a change in the equilibrium position of the rotor. While not necessarily destabilizing, this change in position is undesirable. In order to produce a zero steady-state error, an integrator is added, which increases the system type, or the accuracy and ability to reject disturbances. The design of the controller is then in the selection of the three proportionality constants, k_P , k_I , and k_D , and the time-response constant, τ , which has not yet been discussed.

2.2.C Root Locus Design Methods

Classical control theory offers several techniques for choosing the gains of this controller. One of the most useful control tools is a root locus plot. The magnetic bearing plant transfer function (equation (2.17)) consists of a denominator, which is a polynomial function of s , and a numerator of some lesser order—a simple scalar in this case. The behavior of the system can be determined from the location of the roots of the characteristic equation, which results from equating the denominator of the transfer function to zero. As discussed above, a system with poles in the right half of the complex plane is unstable. The poles of the closed-loop control system, or system with feedback, can be determined from the poles and zeros of the open-loop system, or the system without feedback, as well as the gain of the controller. The various possible poles can be plotted on a graph as a function of the gain, in what is known as a “root locus plot”. As can be seen in figure 10, the uncontrolled system is unstable for any gain—that is, it always has at least one pole in the right half plane. By adding a controller, the system can be made stable for some gains. The PID controller has the following transfer function:

$$\frac{V_c(s)}{V_r(s)} = \frac{k_D \cdot s^2 + k_P \cdot s + k_I}{s \cdot (s + 1/\tau)} \quad (2.19)$$

V_c : Error voltage between reference and actual position

The poles of the controller are largely preset. One must be at zero, and the other should be far into the left-half plane. The purpose of the second controller pole is to create a “proper” transfer function. A transfer function with a numerator of higher order than the denominator is both undesirable and impossible to construct. Such a system would amplify high frequency signals, which are the domain of random circuit noise. If the signal input to the derivative gain has any significant noise, the derivative output will saturate and likely destabilize the system [3]. The second controller pole is added as part of the derivative gain in the actual implementation, to act as a low-pass filter for the sensor output signal.

The zeros of the controller are a function of k_P , k_I , and k_D . Zeros added to a system have the effect of shifting the root locus to the left, making the system more stable. The selection of zero location for the PID controller is largely a trial-and-error process, using a software package to plot the root locus for various placements of the controller zeros, and then picking the best controller based on frequency response and step response plots for the resulting system. To aid in selection of the controller zeros and gain, a script file was written using the Matlab⁴ software package. The script presents various informative plots for each set of zero locations and gain selected. The text of the script file, `amberroots.m`, is included in Appendix B. The results of a few select trials are included as figures 11-a - 11-l.

Figures 11-a and 11-j show the two general root locus shapes that can be achieved with PID control for this system. Each plot has a stable region, where the gain is large enough to move the unstable poles—the rightmost branches of the locus—into the left-half plane, but not so large as to move either of the branch pairs back into the right-half plane. The location of the poles within the left-half plane indicates several important performance characteristics. The characteristics of poles closer to the imaginary axis dominate the system, since poles farther to the left indicate more quickly decaying exponential functions in the time domain. Many systems can therefore be designed by selection of only one pair of complex poles, whose characteristics describe the behavior of the entire system fairly well. However, for most good pole location choices in this system, four of the five poles are close enough to the imaginary axis to have a significant effect. Therefore, the characteristics of two dominant pole pairs must be considered.

System characteristics that may be found from pole locations include the natural frequency of the system, the damped frequency, the damping ratio, and step input response characteristics such as rise time, settling time, and overshoot. Since the robustness of the magnetic bearing system is of more importance than its step response, the damping ratio was used as the primary design parameter. The damping ratio is equal to the cosine of the angle formed between the negative real axis and a line from the origin through a particular pole, and is indicative of the amplitude of oscillation in a system for given control or disturbance inputs. A high damping ratio, or a pole close to the real axis, is desirable for robust regulation. For the root loci that are possible in the controlled magnetic bearing system, an increase in gain to move one pair of poles closer to the real axis moves the other pair of dominant poles farther away from the real axis. The maximum achievable damping ratio then occurs when both pairs of dominant poles are at the same angle from the origin, as indicated in figure 11-a. After the user has selected the controller zeros and gain in `amberroots.m`, the file calls several Matlab functions to graph characteristics of the controlled system as just designed. The two plots that are created are a step response plot and a frequency response, or Bode plot. Although the

⁴ Matlab is a programmable mathematics software package designed for linear algebra. Through an extensive library of included functions, a great deal of control systems design can be performed in the Matlab environment.

step response of the system is not significant per se, it does show a few important characteristics of the system—most importantly, the damping ratio. If the step response contains fast oscillations, or oscillations of high amplitude, as in figure 11-h, then the damping of the system is unsatisfactory.

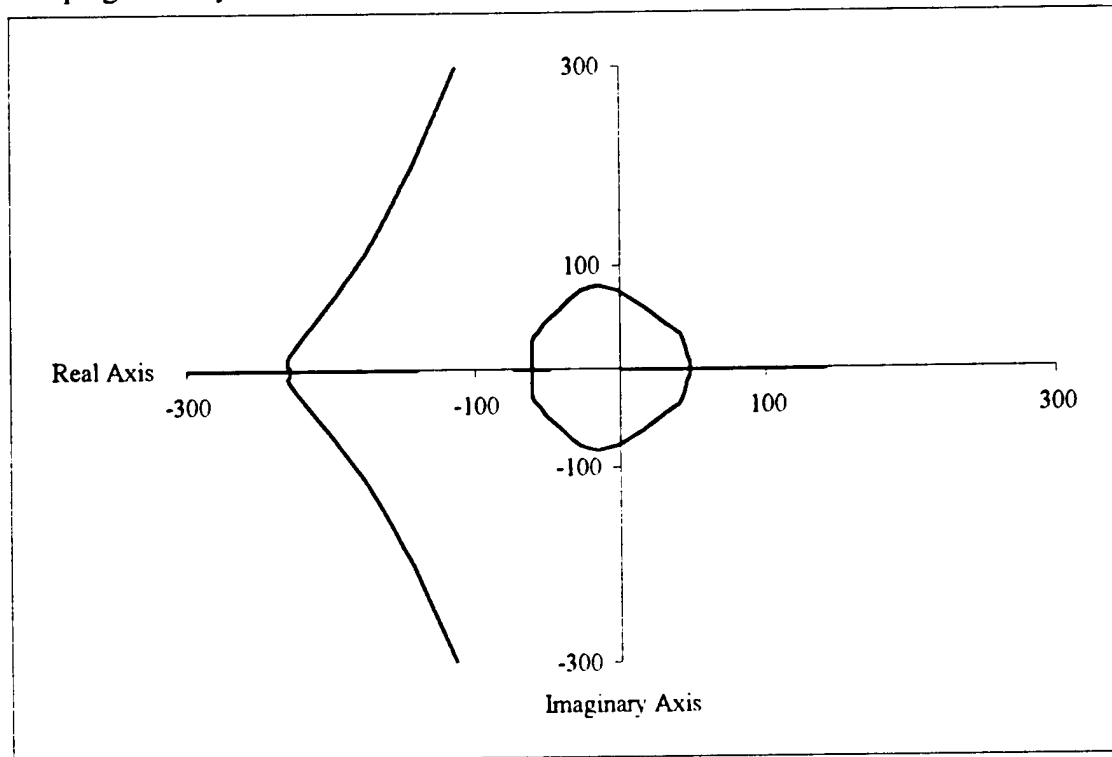


Figure 11-a Root locus of magnetic bearing system with controller zeros at -50 and -75

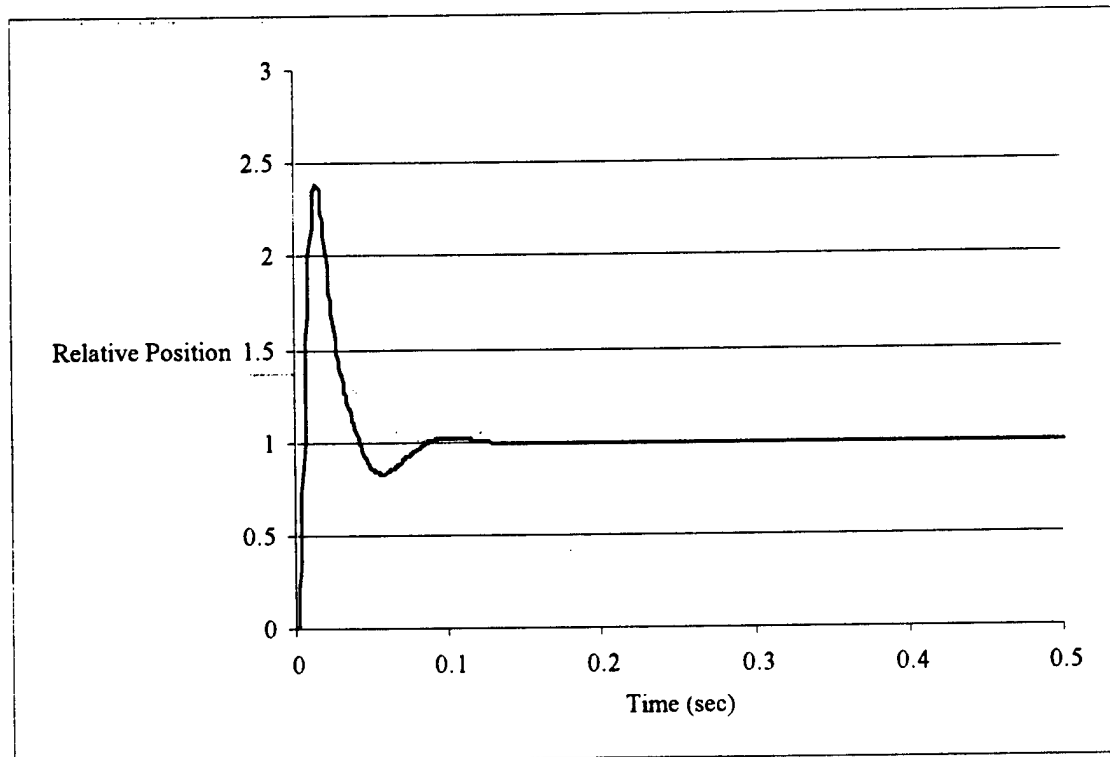


Figure 11-b Step response of magnetic bearing system with controller zeros at -50 and -75

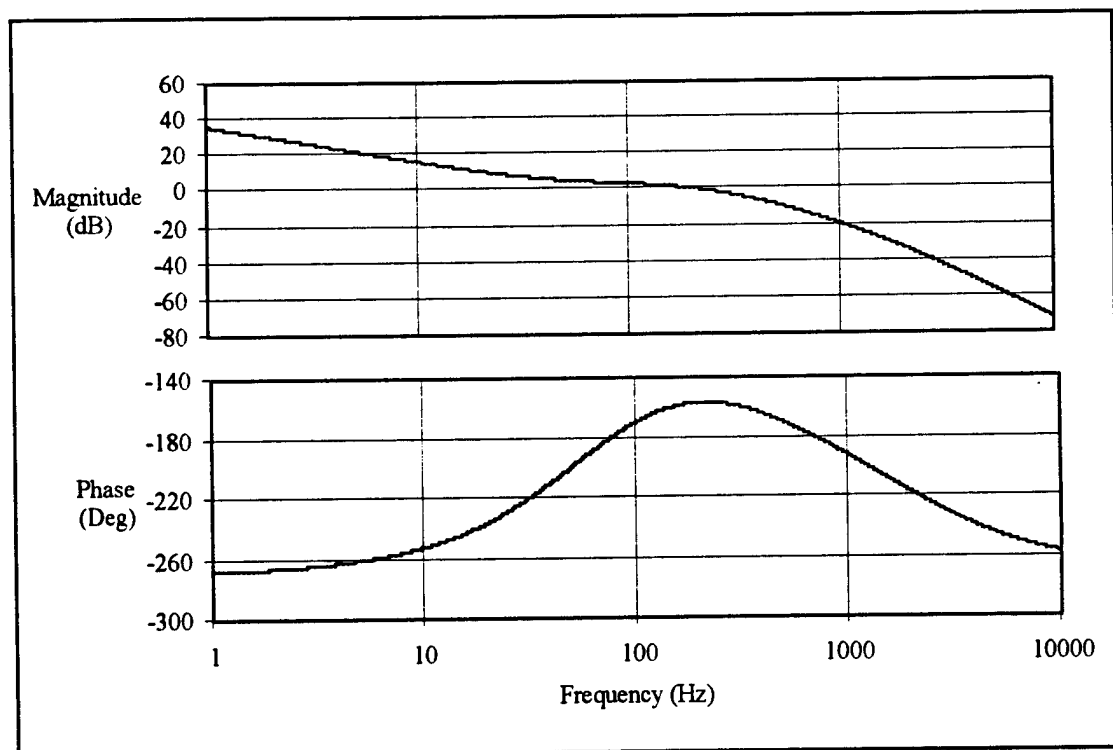


Figure 11-c Frequency response of magnetic bearing system with controller zeros at -50 and -75

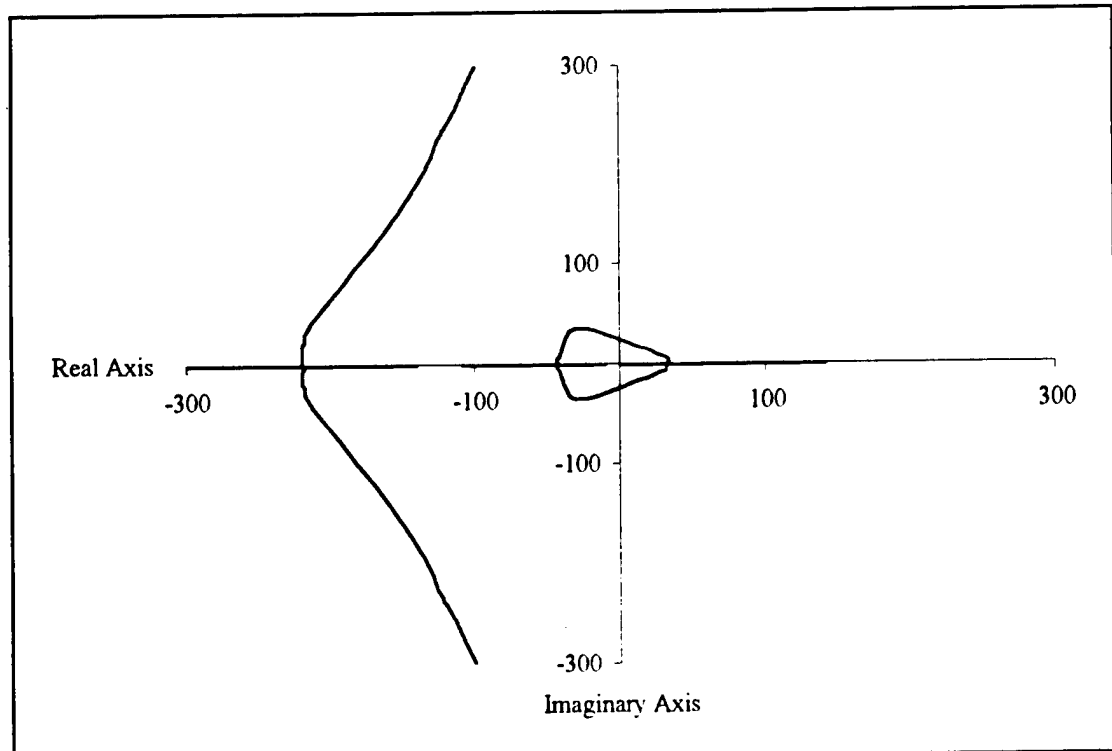


Figure 11-d Root locus of magnetic bearing system with controller zeros at -10 and -110

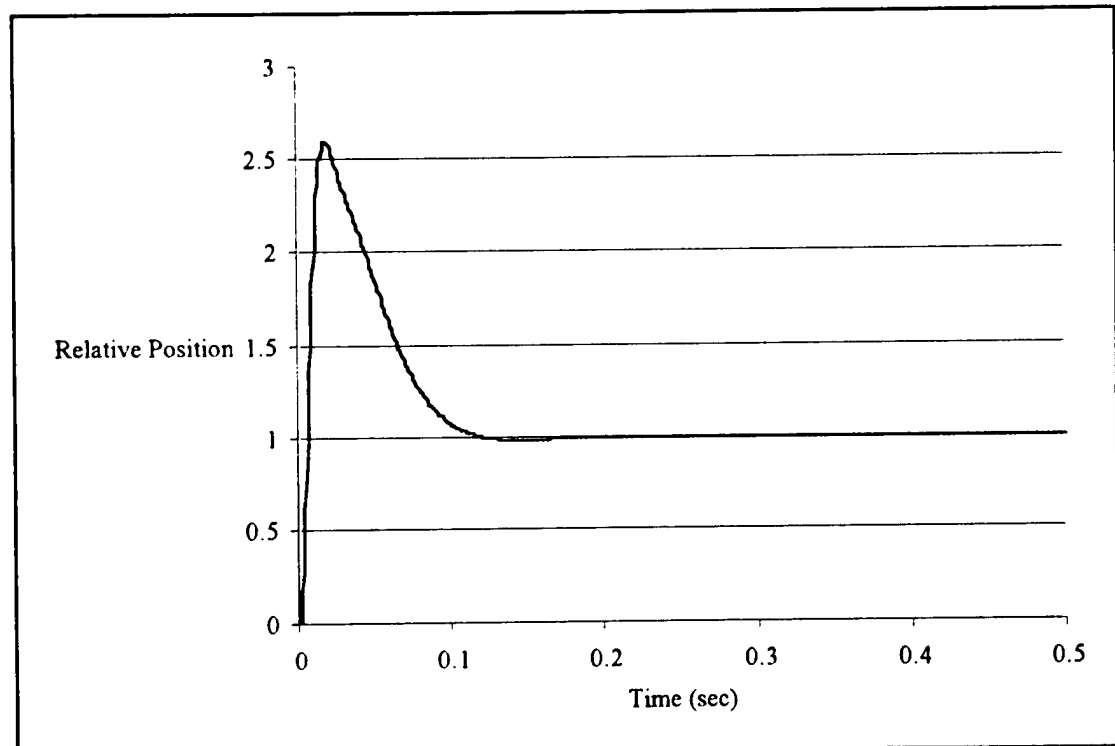


Figure 11-e Step response of magnetic bearing system with controller zeros at -10 and -110

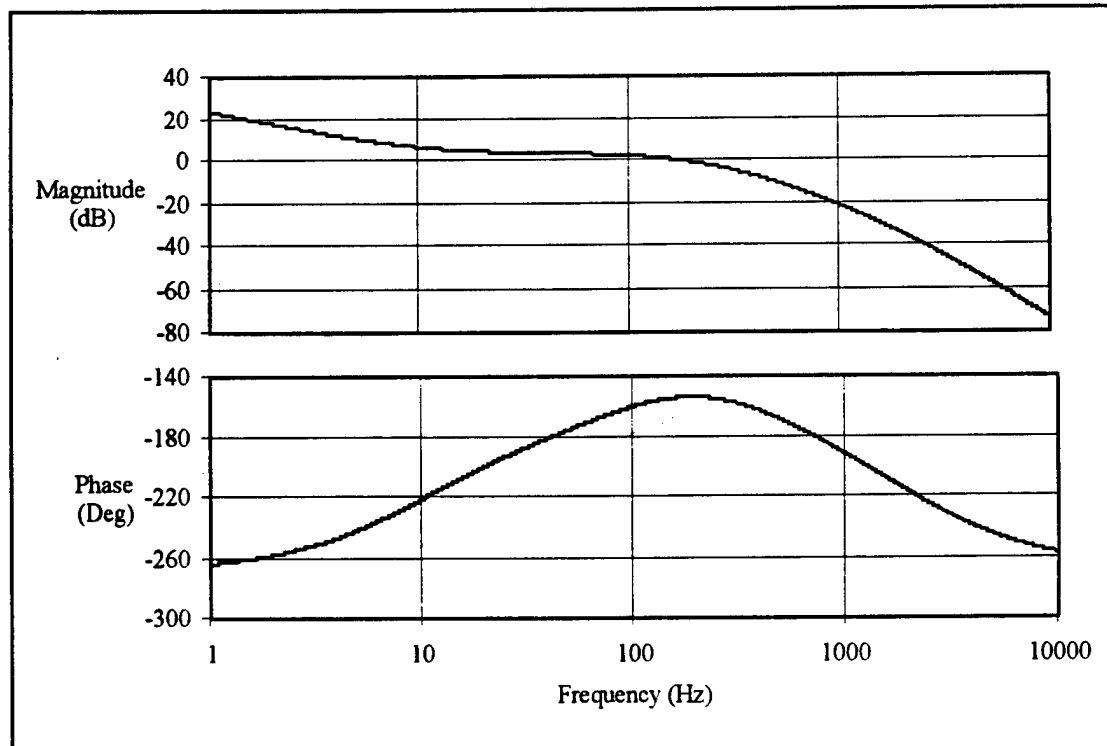


Figure 11-f Frequency response of magnetic bearing system with controller zeros at -10 and -110

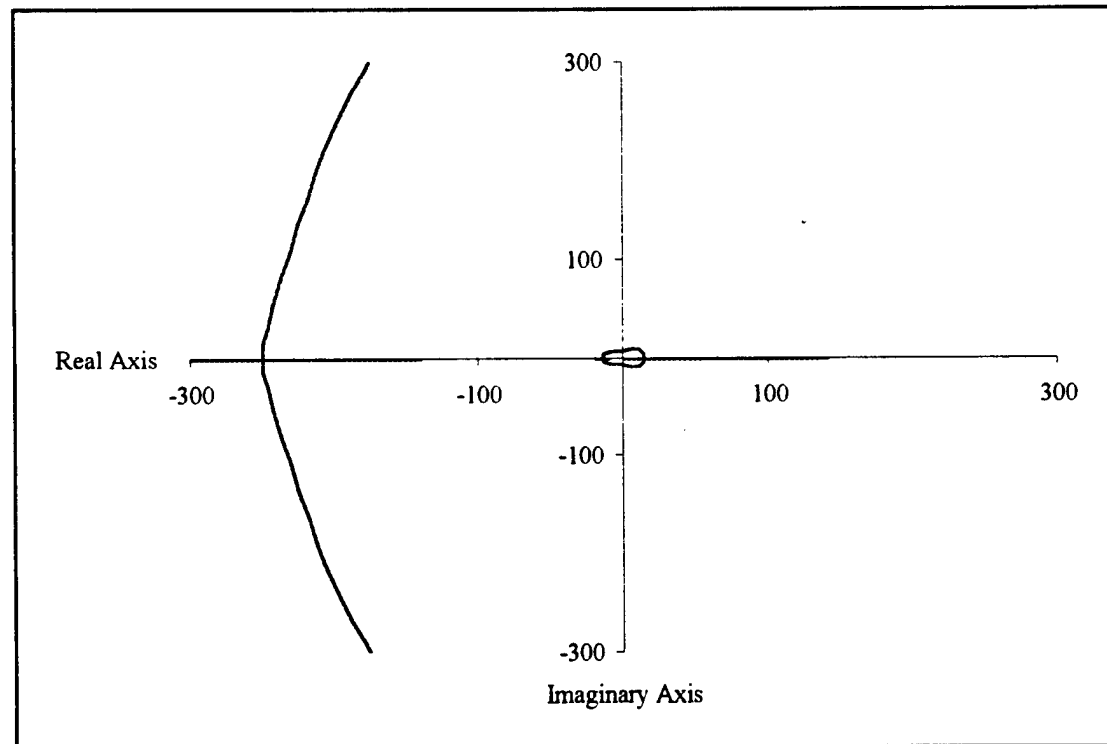


Figure 11-g Root locus of magnetic bearing system with controller zeros at -10 and -20

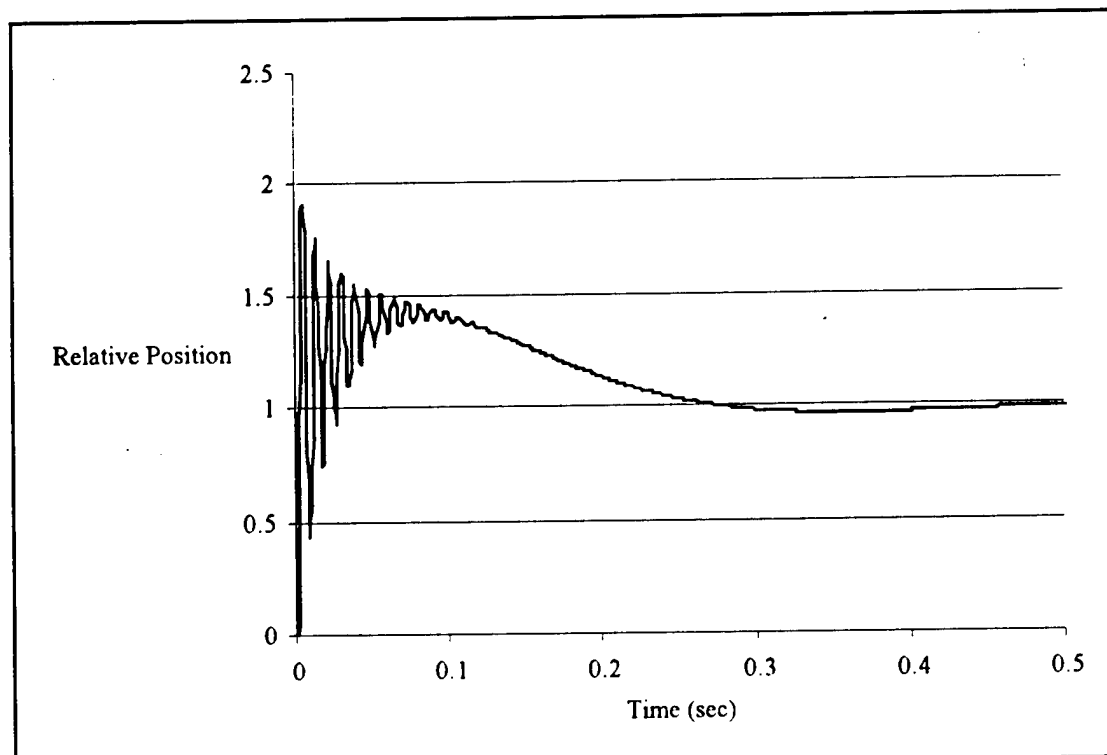


Figure 11-h Step response of magnetic bearing system with controller zeros at -10 and -20

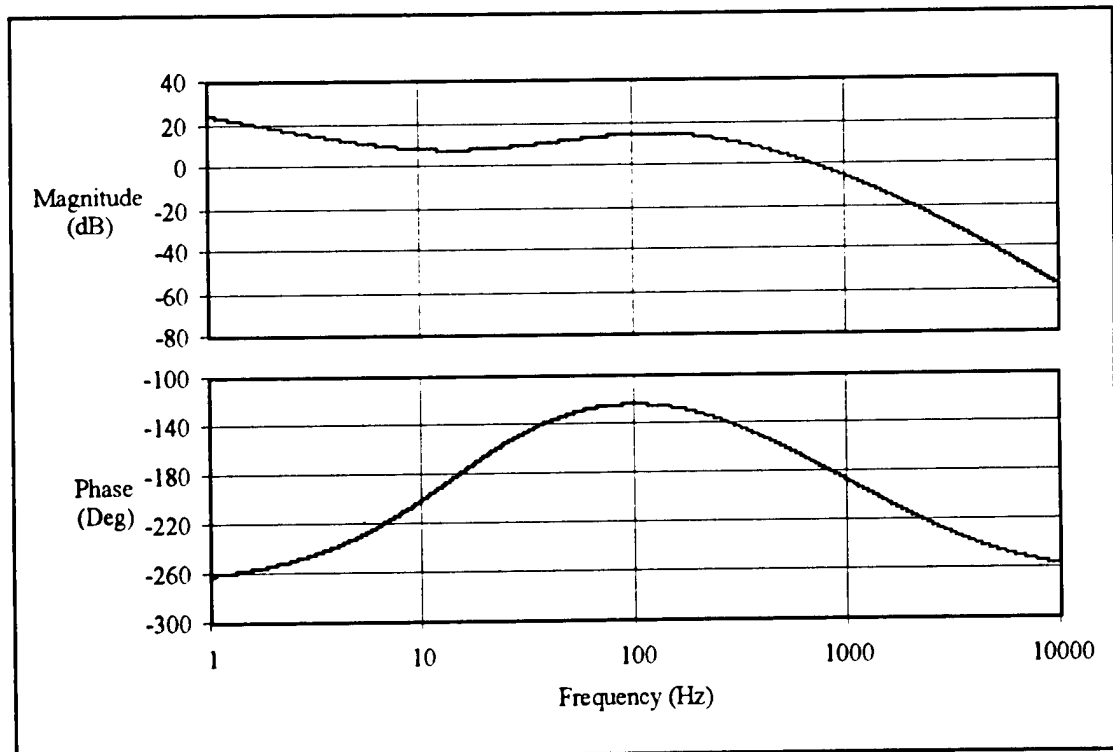


Figure 11-i Frequency response of magnetic bearing system with controller zeros at -10 and -20

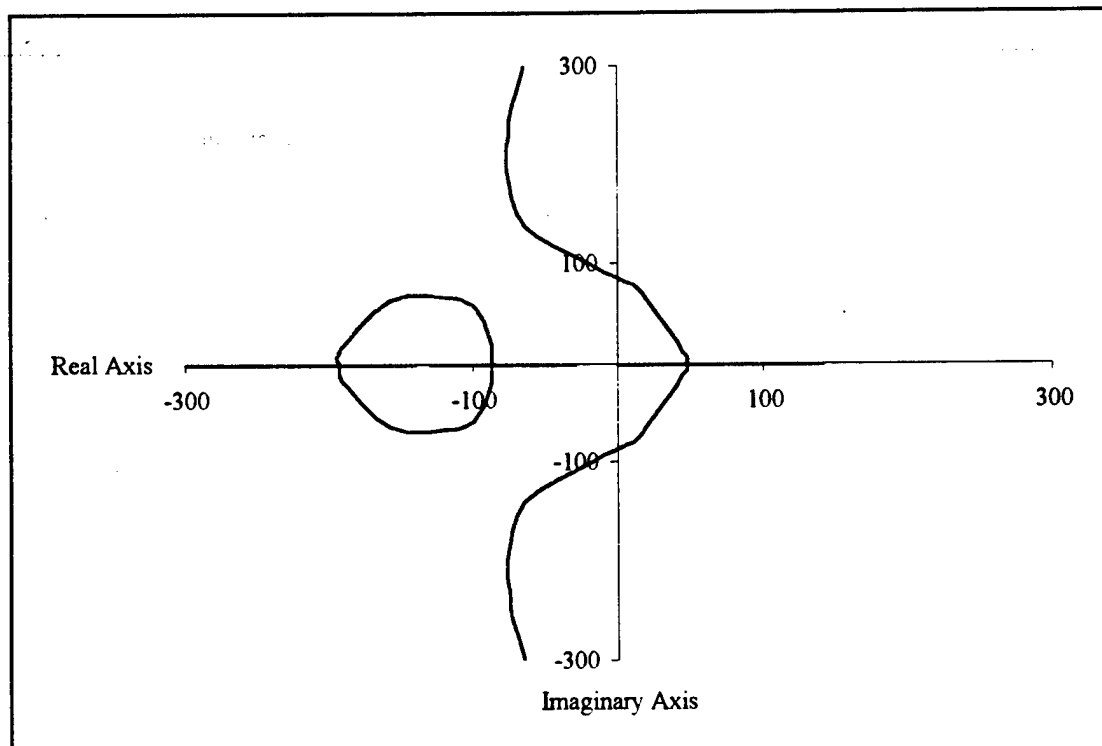


Figure 11-j Root locus of magnetic bearing system with controller zeros at -30 and -130

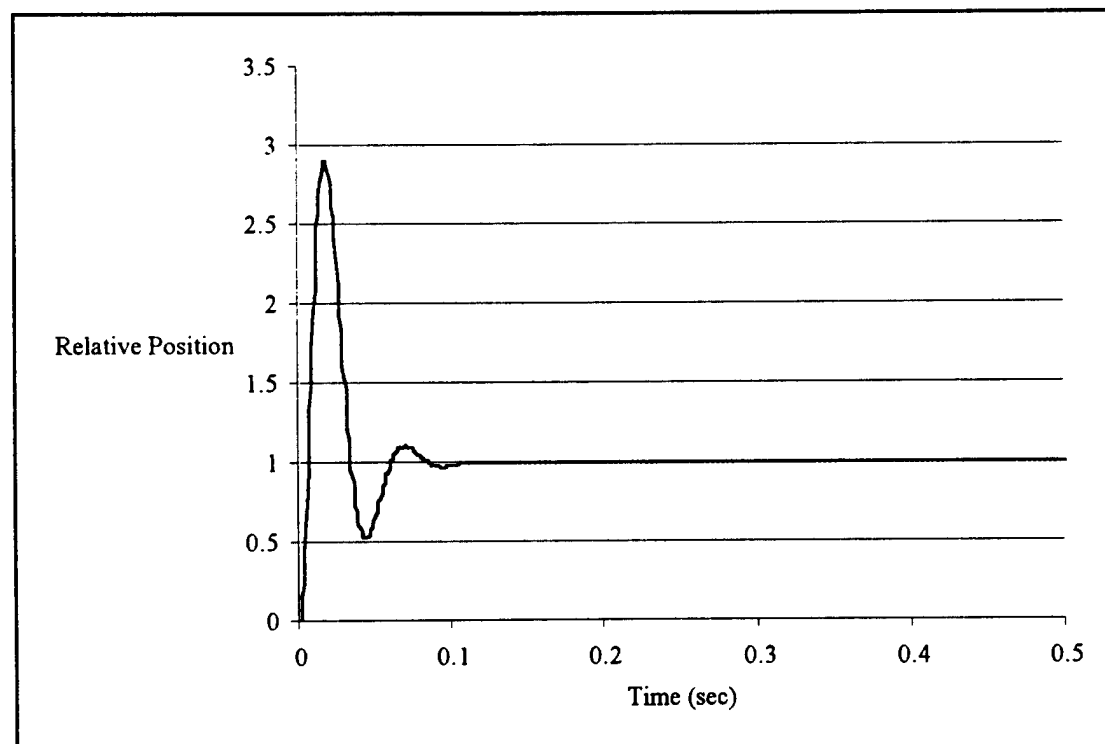


Figure 11-k Step response of magnetic bearing system with controller zeros at -30 and -130

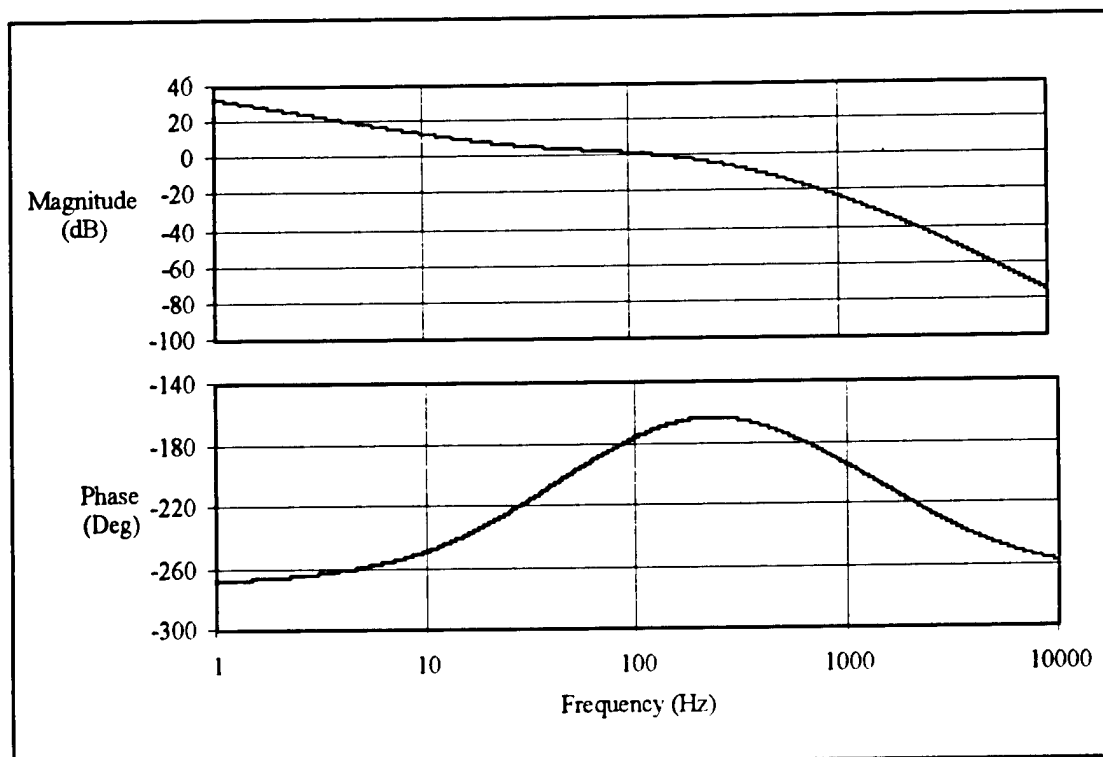


Figure 11-1 Frequency response of magnetic bearing system with controller zeros at -30 and -130

2.2.D Frequency Response Design Methods

When any stable system is driven by a sinusoidal input, it will have an output of identical frequency, but varying phase and amplitude with respect to the input. The Bode plot—the final graph printed by `amberroots.m`—is a graph of this phase and amplitude shift in the output of the system compared to the phase and magnitude of a sinusoidal input, plotted for a range of frequencies. Since the amplitude of the response can be extremely large or small, and the range of frequencies is quite large, the Bode plot uses a logarithmic scale for frequencies and gains. Gains are plotted in decibels. Using a logarithmic scale for amplitude has the added advantage of making the plots of two systems in series additive. Thus, the frequency response of the bearing plant can be added to the controller frequency response to arrive at the plot of the total open-loop system.

The stability of most systems can be determined from the Bode plot⁵. When the phase shift of the system is 180° , the gain must be less than 0 dB in order for the system to be stable. The amount of change that a system can undergo before reaching instability by this criterion provides a measure of the “relative stability,” or robustness of the system. The amount of phase that could be added to or subtracted from the system before it would become unstable—before it would reach 180° at the 0 dB gain crossing

⁵ This applies to minimum-phase, linear, time-invariant systems, such as the magnetic bearing system.

point—is known as the phase margin. Conversely, the amount of gain that could be added to the system before causing it to become unstable, or reach 0 dB gain at the 180° phase crossing, is known as the gain margin. Gain and phase margin are both indicative of the robustness of the system. Phase margin is of greater importance in most systems, since the gain of the system can usually be measured with more certainty. The controller design objective from a frequency response standpoint, therefore, is to maximize the phase margin and ensure that the gain margin is not dangerously small.

As mentioned above, the Bode plots of a system are additive. The frequency response of the uncontrolled magnetic bearing plant is shown in figure 12-a. The controller has two poles, one at the origin, and one far to the left, whose frequency responses combine with the plant to give the response shown in figure 12-b. The design parameters in a frequency response method are primarily the zeros of the controller, as was the case with root locus. The zeros add positive phase to the plant, as shown in figure 12-c, creating a “phase bubble” that will push the system phase over 180° at the 0 dB gain crossover frequency. The results of a few zero placements are shown in figures 11-c, 11-f, 11-i, and 11-l. While the zero in figure 11-i provides the largest phase bubble, the bubble is not centered on the 0 dB gain crossover, and the phase margin is thus not especially large. The plot in figure 11-f shows the largest phase margin, but a dangerously low gain margin. The system shown in figure 11-c has the best combination of phase and gain margin. The phase margin is not large—only 24° —but seems to be the best available.

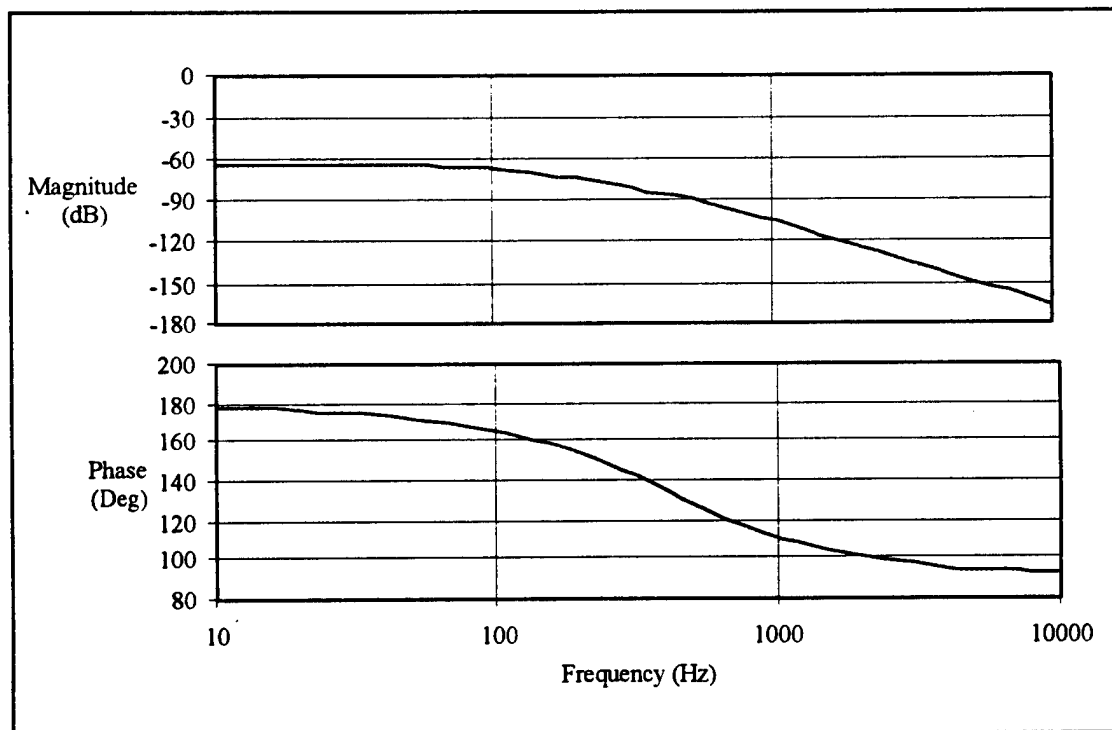


Figure 12-a Frequency response of uncontrolled magnetic bearing plant

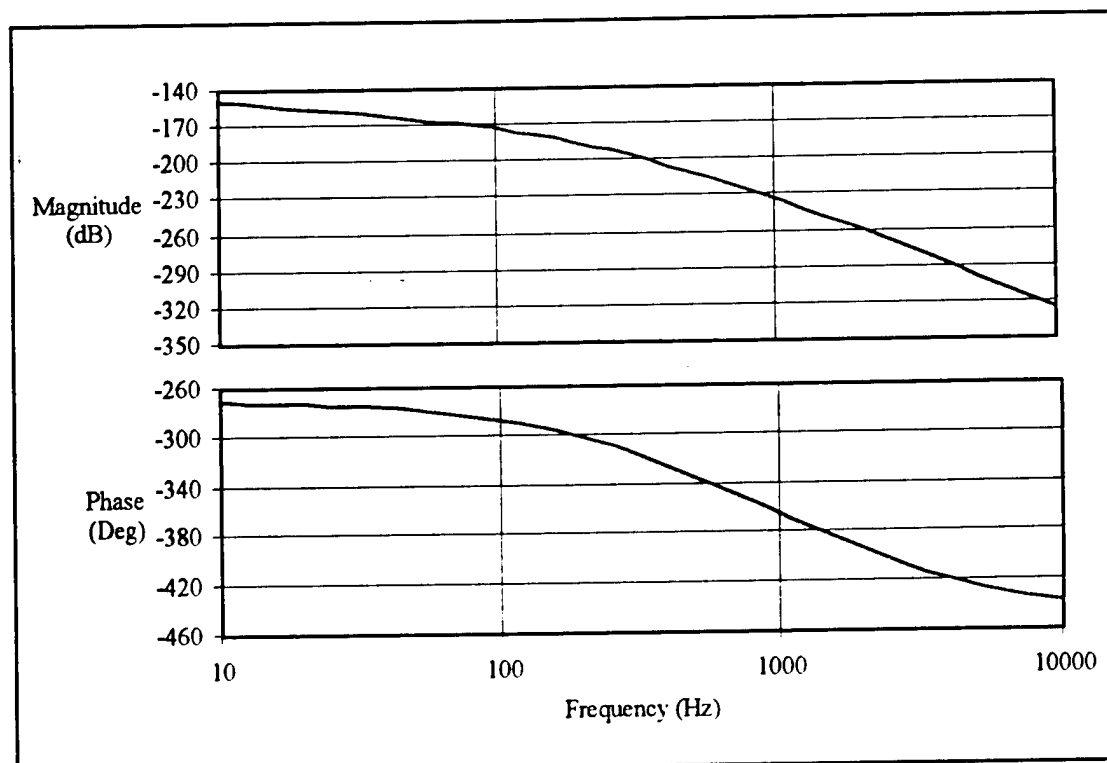


Figure 12-b Frequency response of magnetic bearing plant and controller poles

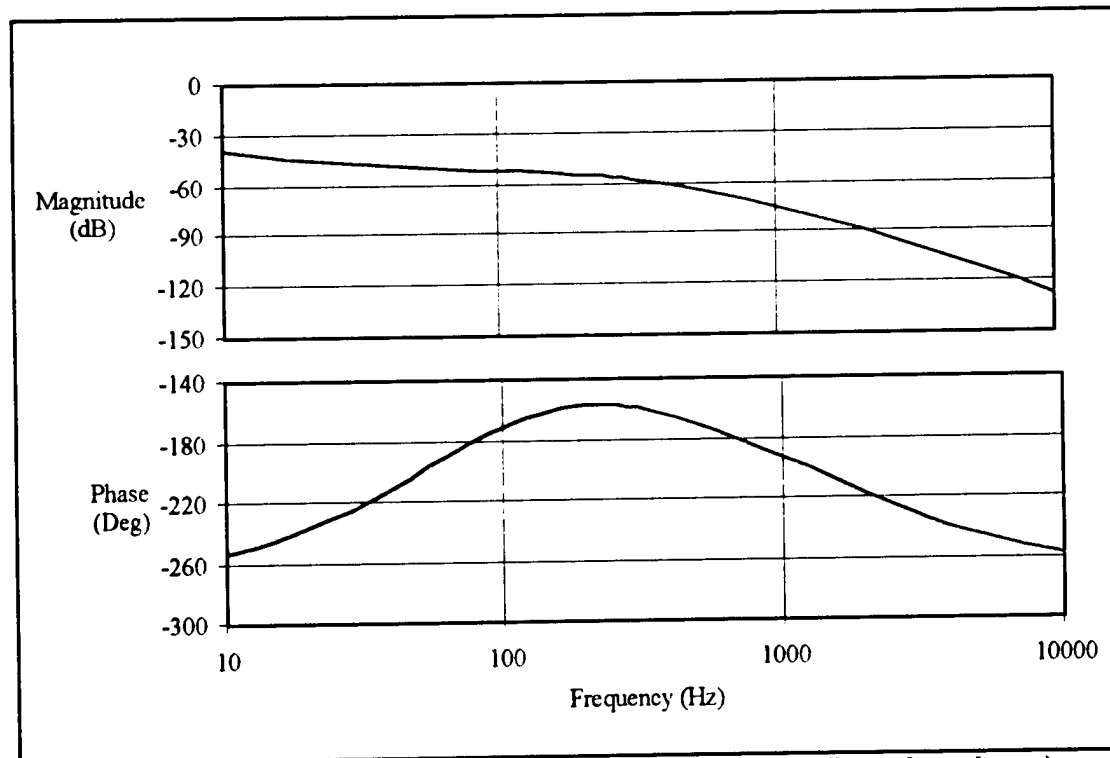


Figure 12-c Frequency response of magnetic bearing plant and full controller (poles and zeros)

2.3 IMPLEMENTATION

2.3.A Controller

The desired mathematical transfer function of the controller was given in equation (2.20). This transfer function was realized as an electrical network based on operational amplifiers. The circuit diagram for the controller is shown in Appendix C. This form of the controller is more accurately described by the following transfer function:

$$\frac{V_c(s)}{V_e(s)} = k_p + \frac{k_i}{s} + \frac{k_d \cdot s}{s + 1/\tau} \quad (2.20)$$

The only mathematical difference between the two transfer functions is the value of the constants, but practical differences arise due to the design of the circuit. As indicated in the circuit diagram, a separate op amp is used for each of the three components of the controller—proportional, derivative, and integral—with a fourth amplifier to sum the outputs of the other three and provide an overall gain. While it is possible to implement the above transfer function with only two op amps, the design chosen provides good separation of the proportional, integral, and derivative components, so that characteristics of each may be altered without affecting the others. The output of the derivative amplifier tends to have large oscillations if a sensor signal is input to it directly, since the signal will inevitably contain noise. Signal noise generally occurs at high frequencies, so the derivative of the noise is very large. Therefore, the signal must be filtered to improve the quality of the derivative output. This filtering is done by including a low-pass filter in the derivative amplifier. The time constant τ determines the corner frequency of the filter, or the maximum frequency for which the filter passes more than half of the power in a signal.

2.3.B Amplifiers

The output of the control circuitry cannot be used directly to drive the bearings, as the op amps in the circuit are only capable of handling current on the order of milliamps. The bearings, on the other hand, may require several amps of current at times, and have an operating point current close to one amp. The output of the controller, then, must be fed to an amplifier circuit. Traditional amplifiers, such as power transistors, have an output voltage that is proportional to the input voltage in a nearly linear fashion, but are inefficient and can overheat easily. A common solution to the related problems of efficiency and overheating is pulse-width modulation. The output voltage of a pulse-width modulating amplifier can have only two values: high and low. The exact high and low voltages are determined by the power supply, but the voltage output of the amplifier will not be anything other than those two values. In order to achieve the same effect as an intermediate voltage, the amplifier switches on and off very quickly, remaining on

slightly longer to output a higher value, and off slightly longer to output a lower value. The time for one complete cycle—on once and off once—is fixed. The output, then, is a square wave with a duty cycle—or percentage of the cycle that the amplifier is providing a high voltage—proportional to the input voltage. The magnetic bearings, which are large inductive circuits, act as low pass filters, averaging the output of the amplifier. The end effect on the system is thus much the same for a pulse-width modulating amplifier as for a linear amplifier. The amplifier itself, though, is much more efficient when using pulse width modulation. The specifications of the power amplifiers used in the experimental apparatus are given in Appendix D.

2.3.C Sensors

The final topic in the design of this control system is the use of sensors. Sensors are necessary to provide feedback, without which effective control of the magnetic bearing system would be impossible. The magnetic bearing controllers use “eddy-current” sensors to detect the position of the shaft. Each sensor is composed of several loops of wire, through which a small AC current flows. The alternating magnetic field produced by the sensor induces eddy currents in any nearby conductors. These currents draw power from the sensor, which can be translated into a voltage loss. The signal conditioning equipment supplied with the sensors performs this conversion, and outputs a voltage proportional to the distance of the shaft from the sensor [22]. This voltage is amplified and shifted so that it is zero at the operating point, and then sent to the controller. The radial bearings each have two sensors—one for the horizontal direction, and one for the vertical. The thrust bearing uses the average of the output of two position sensors for the lateral direction.

There are several considerations in the use of sensors for this application. The output of the sensors shows good linearity and resolution over the full range of shaft motion, but is subject to a significant amount of noise under certain conditions. Since the sensors use magnetic fields, there might be some concern over their performance in such proximity to the strong fields of a magnetic bearing. However, the driving frequency of the sensors is much higher than any frequency in the bearings, so the sensors remain effective. The magnetic bearings are also grounded and driven by high-current amplifiers, which limits the formation of eddy currents. The sensors tend to interfere with one another when placed in the same plane, and are therefore offset slightly in order to yield a cleaner signal. The final concern has not to do with the quality of the sensor signal, but rather with its usefulness. In the design of the controller, the sensors are presumed to measure the position of the bearing rotor. In fact, the sensors are located around the shaft close to the rotor, and thus do not directly measure the position of the rotor. The shaft has a small range of angular motion, so the position of the shaft a centimeter or two away from the rotor is a sufficiently good approximation of the position of the rotor. The position sensors for the thrust bearing, however, are located

near the edges of the thrust rotor, where even a slight rotation of the shaft can cause a significant movement. In order to determine the position of the shaft more accurately, the thrust bearing averages the output of two sensors located at opposite edges of the thrust rotor.

2.4 SIMULATION

Several different simulations of the magnetic bearing system have been created using the VisSim software package, which allows the user to link blocks of various properties together in block-diagram form, and then simulate the behavior of the system. The first simulation constructed was a linear simulation, with most of the system parameters entered as variable blocks, so that the values of various parameters could easily be changed to test for robustness. Other than the changes made to some parameters to test robustness, this linear simulation is equivalent to the transfer functions used in Matlab. The linear simulation provides a quick check on the performance of a controller, and can test robustness to some degree, but is not especially accurate. For final testing before building the controller, a non-linear simulation (Appendix E) was constructed. The non-linear simulation is a model of the behavior of the rotor while constrained in two directions (the horizontal and lateral in the example included), controlled by the magnetic bearings in the third direction, and subject to various disturbances. A separate simulation was constructed for each pair of opposing bearings. The differences between the various channel-pair simulations are minor, consisting mostly of different nominal parameter values and the inclusion of gravity in the vertical channel simulations. The simulated systems are each subdivided into three primary blocks: the controller, the bearing plant, and the sensor.

The sensor block is a simple linear gain, limited in output by the voltage supplies of its signal conditioning circuitry. The value of the gain was determined by experimentation. Sensor noise is simulated by an input to the summing junction of the reference signal and error signal.

The controller is a PID controller, broken down into individual op amps, just as it is actually implemented. The output of each op amp and the final output of the controller are plotted. All op amps are supplied with ± 12 V rail voltages, and thus have an output voltage range of approximately ± 10 V, as determined by experimentation. For the non-linear simulation, therefore, the output of each element of the controller, as well as the total controller output, is hard limited to ± 10 V. This limiting corresponds to the potential saturation of controller op amps in the presence of a large or noisy error signal.

The bearing plant is broken down into three areas: the voltage-current conversion, the current-force conversion, and the force-position conversion. Since this system uses a differential driving mode, both channels have a bias voltage input to the voltage-current conversion block. There is also an input available for an additional bias voltage in the

top channel. This top-only bias voltage would be used to offset the effects of gravity, so that the control signal at equilibrium position could be zero, despite the need for a greater force from the top bearing than from the bottom. In simulation, though, the system behaved best when there was an equal bias voltage on both channels, so the top-only bias voltage input is normally left at zero. The control signal input to the voltage-current conversion blocks is added to the bias voltage in the top channel, and subtracted in the bottom channel, as described above for differential driving. The third voltage input is a "back-emf" generated by the motion of the rotor. Since the rotor is a conductor, and is moving with respect to the magnetic field of the bearings, it generates a voltage across the bearing coils [22]. This induced voltage is related to the velocity of the rotor by a constant, k_u , which is theoretically equal to the k_i introduced as a current-force constant. However, due to losses in the magnetic field, k_u is actually smaller than k_i . The values of k_u used in the simulation are purely guesses, but do not show any significant effect on the system for values less than k_i . The conversion of the sum of the input voltages to a current value follows from the same differential equation derived in the linear plant model. The force-position conversion converts the sum of the forces on the rotor into a position. There are three forces input to this block: the magnetic force from each of the two channels in the direction being simulated, and external disturbance forces. For the vertical direction simulations, the force due to gravity is included as an external disturbance. Other potential disturbances include thrust from the attached fan blade or propeller and forces due to the rotation of the shaft. The sum of the forces is divided by the effective mass of the rotor to determine acceleration of the rotor. The effective mass of the rotor is the inertial load as seen by the forces at the rotor—since forces at one of the end rotor do not accelerate the entire shaft, the "apparent mass," or inertial load, is less than the total mass of the shaft. This inertial load is dependent upon the position of the entire shaft, but remains relatively constant due to the small deviations in position possible for the shaft. The equilibrium inertial load was determined by measuring the force due to gravity on each of the radial bearing rotors while the shaft was at equilibrium position, and dividing by the acceleration due to gravity. The total forces on the rotor divided by the apparent mass yield the acceleration of the rotor. The acceleration is then integrated twice to find the position of the rotor.

The primary plot of interest in the simulation is the rotor position plot. In the vertical channel simulation, the rotor starts from rest on the bottom of the bearing. For a stable and robust system, the position of the rotor approaches zero, or the center of the bearing, as the simulation progresses. The controller selected for implementation behaved quite robustly for this simulation in the presence of large parameter variations and disturbance forces, as shown in the plots included in Appendix E.

2.5

BEHAVIOR OF THE SYSTEM

The first test of the magnetic bearing system was to support the shaft with a bushing in the rear, and a single radial bearing in the front. With minor tuning of the overall controller gain, the magnetic bearing was able to hold the rotor in a fixed position. The rear bushing was then replaced with another radial bearing, which had a somewhat destabilizing effect, as the entire shaft was then free to move some amount in every direction. Again, with minor tuning and coordination of the front and rear bearing gains, the shaft was stabilized in the vertical and horizontal directions. Finally, the thrust bearing was added. While the thrust bearing provided stabilization in the third degree of freedom for the shaft, the thrust rotor, or flywheel, also added a significant amount of weight to the system. Since the bearing controllers do not use a "top-only" bias current to counter the weight of the shaft, the controller output must account for the weight. When the shaft is stable in its nominal position, the proportional and derivative components of the controller have a zero output voltage. The force to counter gravity is then calculated entirely by the integral portion of the controller, which essentially serves as an online calculation of the necessary bias voltage to lift the shaft. With such a heavy shaft, the output of the integrator element of the controller would saturate without fully centering the rotor. In order to remedy this problem, the rail voltages to the controller were increased, as was the gain on the integrator element of the controller. With these modifications, the magnetic bearings were able to stabilize the shaft in a zero-error position.

The system is robust to controller parameters: values of the component gains within 20% of the nominal value do not destabilize the system. Simulations and experiments both indicate robustness to the values of many physical parameters, such as coil inductance, relative permeability of the bearing material, and back-emf voltage, which are all difficult to determine accurately. The parameters to which the bearings are most sensitive are bearing coil resistance and shaft weight, both of which can be measured with a high degree of accuracy. The bearings effectively maintain shaft position even at high rotational speeds—the shaft has been spun up to 3000 rpm, supported entirely by magnetic bearings, without any instability. External forces, such as striking the shaft, can cause the rotors to touch the bearings, but the rotors will return to their nominal position immediately afterwards, indicating that the system can effectively recover from a large disturbance. The magnetic bearing system was determined to meet its initial design specifications, and to meet the requirements of the active noise control system.

3 ACTIVE SOUND CONTROL

There are many different types of noise control systems, most of which operate around one of two principles: feedback and feedforward. A feedback system is the simpler of the two, but is limited in its effectiveness over large areas. Feedback systems are usually used to quiet small regions at some distance from the primary noise source. Feedforward techniques, on the other hand, are more complex, but can be effective over a broad area. Since the goal of this project is to achieve global noise control, a feedforward technique was selected. In order to maximize the effectiveness of the noise control system, and to increase its robustness, the system was designed to be adaptive—able to adjust itself to an optimal state.

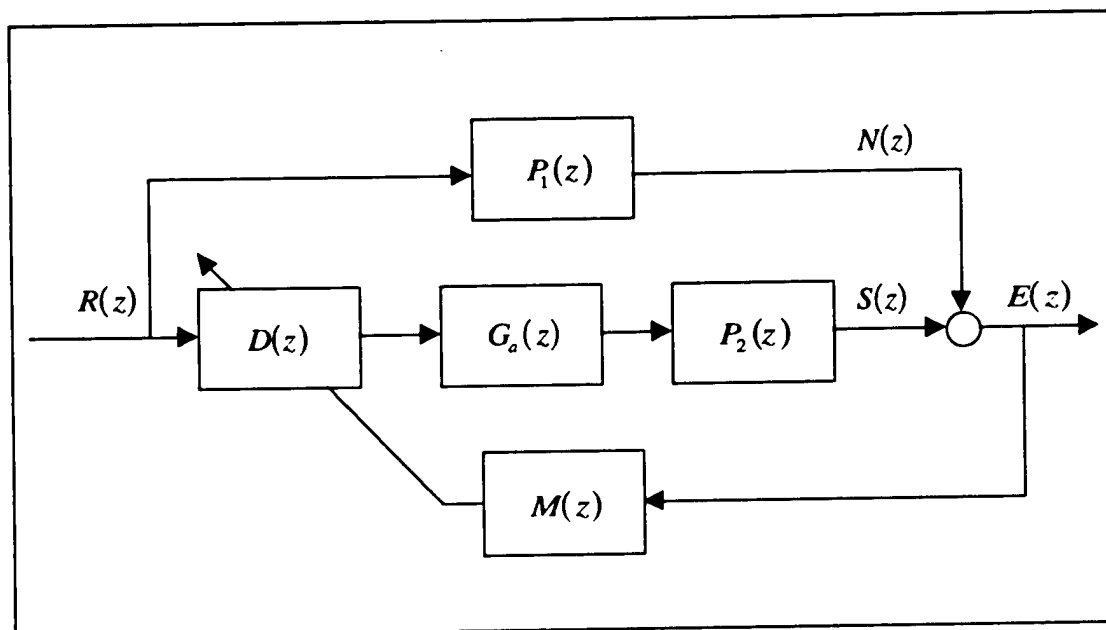


Figure 13 Adaptive feedforward active noise control system block diagram

Mathematical analysis of the noise control system is done using different techniques than were used to develop the magnetic bearing system. A central tool used in the magnetic bearing system analysis was the Laplace transform, which converted time-domain information into the frequency domain; the resulting signals and transfer functions were represented as functions of the complex variable "s." The Laplace transform worked well for the magnetic bearings, which are a continuous-time system. However, use of the Laplace transform becomes more awkward in the discrete-time systems that arise in digital control⁶. Since the noise control system involves a digital signal processor, which is a discrete-time component, a mathematical tool known as the

⁶ The value of a discrete-time system or signal can change only at certain intervals, as opposed to the continuously varying values of analog signals.

"z-transform" is used for analysis. The z-transform is derived from the Laplace transform, but is better suited to use in discrete-time systems. Transfer functions and signals derived through use of the z-transform are represented as functions of the complex variable "z." Time-domain signals in discrete systems are represented as functions of "n," which represents a particular sample period.

A complete adaptive feedforward noise control system is shown in block diagram form in figure 13. The objective of the system is to reduce the amplitude of a sound present in the air. In this project, the primary sound—whose amplitude is to be reduced—results from the normal operation of the motor and fan blade. The sound travels through a path in the air—which is modeled as the transfer function $P_1(z)$ —and arrives at the error microphone. In order to reduce the amplitude of the primary sound, the noise control system generates another sound—the secondary sound—at the same frequency as the primary sound, but exactly out of phase. The two sound waves will destructively interfere with one another, reducing the amplitude of sound at that frequency. In order to produce the secondary sound, the noise control system must have some reference to the frequency of the primary sound. For this project, the reference signal, $R(z)$ is generated from a tachometer, since the frequency of one of the loudest sounds produced by the motor and fan blade is proportional to the frequency of rotation of the fan blade. The reference signal, or feedforward signal, passes through a digital filter, $D(z)$, which adjusts the phase so that the secondary sound is out of phase with the primary sound. The output of the filter is converted to a sound by an actuator, represented by the transfer function $G_s(z)$. This actuator is often nothing more than a speaker, though in the case of this project it is the magnetic bearing system. This secondary sound travels through a path in the air, $P_2(z)$, and arrives at the error microphone. The primary and secondary sounds combine additively at the microphone, as at any other point in space, yielding a total sound that is measured by a microphone. The noise control system seeks to drive this total sound to zero, or as small an amplitude as possible.

3.1 ADAPTIVE DIGITAL FILTERING

Adaptive feedforward control involves predicting the noise that the primary source will produce by filtering the feedforward signal, comparing the results of the prediction—in the form of the signal from the error microphone—and adjusting the filter to correct the error. The feedforward signal, which has the same frequency as the primary noise, is used by the controller to produce the secondary sound. The controller adjusts the reference frequency in phase and amplitude using an adaptive filter. The filter output is then used to drive the actuator and produce the secondary sound. A sensor measures the final output noise level and feeds this information back to the adaptive

filter, which adjusts itself to minimize the total sound in the air. The heart of the noise control system is the adaptive filter.

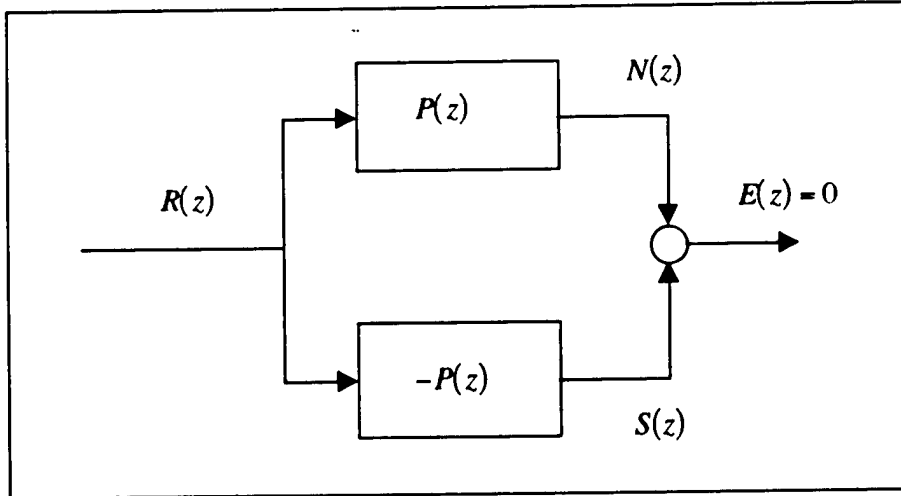


Figure 14 Ideal final state of an active noise control system

An adaptive filter contains two parts: the filter and the adaptation algorithm. In developing the desired form of these two elements, the entire system must be considered. Ultimately, the objective of the system is to achieve complete cancellation of the primary and secondary sound. This can be achieved by a system in which the overall primary and secondary paths are negatives of one another, as shown in figure 14. The error signal is simply the sum of the primary and secondary sounds, which, in this case, yield zero.

$$E(z) = N(z) + S(z) \quad (3.1)$$

$E(z)$: Error signal

$N(z)$: Noise signal, or primary sound

$S(z)$: Secondary sound

As can be seen from the block diagram, the primary and secondary sounds are both functions of the reference signal:

$$N(z) = R(z) \cdot P_1(z) \quad (3.2)$$

$$S(z) = R(z) \cdot G_c(z) \cdot G_a(z) \cdot P_2(z) \quad (3.3)$$

$R(z)$: Reference signal

$G_c(z)$: Controller (filter) transfer function

$G_a(z)$: Magnetic bearing actuator transfer function

$P_1(z)$: Path transfer function from primary sound to error microphone

$P_2(z)$: Path transfer function from secondary sound to error microphone

The error signal, which is desired to be zero, can thus be given as a function of the reference signal:

$$E(z) = R(z) \cdot [P_1(z) + G_c(z) \cdot G_a(z) \cdot P_2(z)] = 0 \quad (3.4)$$

Since the primary and secondary sound sources are nearly identical, the two paths are assumed to have the same transfer function, which yields:

$$E(z) = R(z) \cdot P(z) \cdot [1 + G_c(z) \cdot G_a(z)] = 0 \quad (3.5)$$

This equation shows that, as in the simplified system of figure 14, the objective is to create a secondary path transfer function which is the negative of the primary path.

$$1 + G_c(z) \cdot G_a(z) = 0 \quad (3.6)$$

Since the actuator transfer function is fixed, the filter is set to cancel the actuator transfer function, producing a total secondary path transfer function which is the negative of the primary path:

$$G_c(z) = -\frac{1}{G_a(z)} \quad (3.7)$$

The necessary controller transfer function is the negative inverse of the actuator transfer function. This seems to be fixed, which brings into question the need for an adaptive filter: if the actuator transfer function were known, or could be accurately approximated, use of an adaptive filter would be unnecessary. However, an adaptive filter provides all the same benefits of feedback in a control system: the filter is robust to parameter variations and inaccuracies in the actuator model, as well as differences in the paths that were assumed to be the same.

The filter employed is a finite impulse response filter, meaning that the output of the filter will go to zero in a finite amount of time if it is excited by an impulse input, or that the filter output is a function only of past inputs to the system. The filter transfer function has the following form:

$$c(m) = w(0) \cdot r(m) + w(1) \cdot r(m-1) + \dots + w(n) \cdot r(m-n) \quad (3.8)$$

$c(m)$: Filter output at m^{th} discrete time period

$w(n)$: n^{th} filter coefficient, or weight

$r(m)$: Filter input at m^{th} discrete time period

Which is equivalent to the z-domain function:

$$C(z) = w \cdot R(z) \quad (3.9)$$

$C(z)$: z-domain filter output function

$w(n)$: Weight vector of length n

$R(z)$: Input signal vector containing the last n samples

As shown above, this filter should approximate the negative inverse of the actuator transfer function. The actuator is technically an infinite impulse response system, meaning that the output value after an impulse input is of infinite length, although it approaches zero as time progresses, provided the system is stable. The inverse of the actuator transfer function also has an infinite-length impulse response, so the filter can only approximate the true value of the inverse. In order to get an accurate approximation, the length of the impulse response of the filter should be as long as the significant portion of the impulse response of the actuator, which is commonly held to be the length of time for which the value of the output is greater than two percent of the input. For a finite impulse response digital filter, the length of the impulse response can be adjusted through either the sampling time or the length of the weight vector. The length of the impulse response of the actuator was determined by simulation, and the filter sampling time and weight vector were adjusted appropriately.

3.1.A Least Mean Squares

With the form of the filter itself determined, the next step is to create an algorithm for adapting the weight vector to an optimal form. The optimal form of the weight vector would yield a minimum error magnitude, or a minimum squared error, ζ . The derivation that follows is adapted from [12] and [28].

$$\zeta(n) = [e(n)]^2 \quad (3.10)$$

The error is the difference between the desired output and the actual output:

$$e(n) = d(n) - y(n) \quad (3.11)$$

The actual output is the product of the weight vector and the input vector:

$$y(n) = \mathbf{w} \cdot \mathbf{x}(n) \quad (3.12)$$

$$\therefore e(n) = d(n) - \mathbf{w} \cdot \mathbf{x}(n) \quad (3.13)$$

Since the desired output and the input vector are fixed, the squared error is a function of the weight vector. A graph of $\zeta(n)$ against the possible weight vectors, \mathbf{w} , would yield a paraboloid, with a minimum error at some value of the weight vector. The objective of the adaptive portion of the filter is to shift the weight vector so that it approaches this optimal value. One of the most straightforward and effective methods of adapting the weight vector to arrive at the minimum error is the "Least Mean Squares" (LMS) method, developed by Widrow around 1960. The procedure for finding the optimal weight vector with the least mean squares algorithm, given any initial weight vector, is to move in the direction of the negative derivative of the squared error function—toward the

smallest value of the squared-error paraboloid. The derivative of the error function is found by differentiating equation (3.10):

$$\frac{d\zeta(n)}{dw} = \frac{d\zeta(n)}{de(n)} \frac{de(n)}{dw} \quad (3.14)$$

$$\frac{d\zeta(n)}{de(n)} = 2 \cdot e(n) \quad (3.15)$$

$$\frac{de(n)}{dw} = -x(n) \quad (3.16)$$

$$\therefore \frac{d\zeta(n)}{dw} = -2 \cdot x(n) \cdot e(n) \quad (3.17)$$

Equation (3.17) indicates the slope of the error surface with respect to the weight vector. In order to change the position on the error curve for the next sample, the weight vector must be shifted some amount in the direction of the negative of the above derivative:

$$w(n+1) = w(n) + \mu \cdot x(n) \cdot e(n) \quad (3.18)$$

where μ is the step size, which determines performance characteristics such as stability of the algorithm and convergence time.

The final LMS algorithm is shown in block diagram form in figure 15. The only parameters of the LMS algorithm that are available for change are the convergence step size, μ , and the initial state of the weight vector. However, there are still several factors that affect the behavior of the filter independently of the adaptation algorithm, such as the number of taps in the weight vector and the sampling time of the system. The determination of values for these parameters is discussed later.

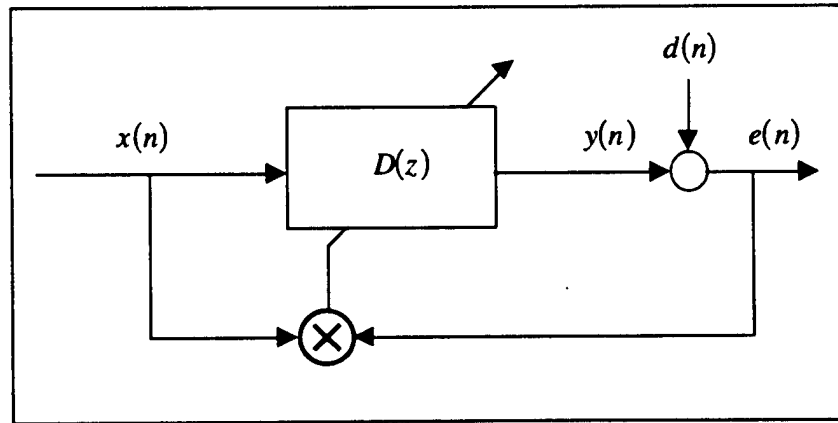


Figure 15 Block diagram of the LMS algorithm

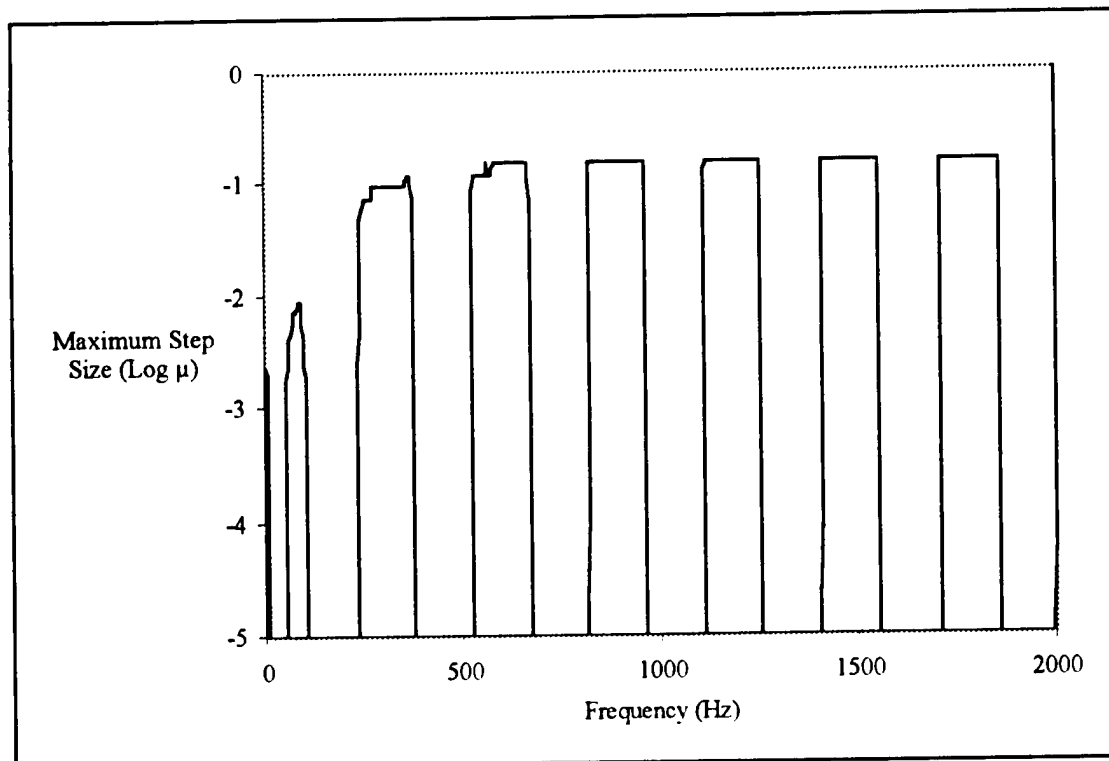


Figure 16 Stability of the LMS algorithm. The graph shows the largest convergence step size (on a logarithmic scale) for which the algorithm will converge. Regions with no plotted values are unstable at any convergence step size.

3.1.B Filtered-X

The sound control algorithm for this project is used in a real, physical application, with outside disturbances and signal noise present. Measurements in the system, especially those of the error signal, are subject to these inaccuracies. The LMS algorithm relies on the measurements not to provide a perfectly accurate error measurement, but to provide an unbiased estimate of the error. So long as the statistical distribution of measurements is centered on the true error value, the estimate of the gradient used in the LMS algorithm will be accurate, and the weight vector will converge toward an optimal value. If, on the other hand, the error measurements used in calculating the gradient are biased from the true error, the algorithm may not converge. One feature of most noise control systems that can cause the error measurements to be biased is the presence of additional “secondary path” transfer functions—that is, additional transfer functions after the filter $D(z)$ in figure 13. While the LMS algorithm relies on the adaptable filter to control the phase of the secondary sound, these secondary path transfer functions each cause additional phase shift in the secondary sound. This phase shift serves to bias the error measurements taken from the error microphone, and, if sufficiently large, can cause the noise control system to become unstable. This can be seen in simulation, such as the bands of instability that appear in figure 16. When the phase shift of the secondary path

transfer functions is less than 90° , the system is stable, and will converge for relatively large μ values. When the phase shift is greater than 90° , the system does not converge for any value of μ .

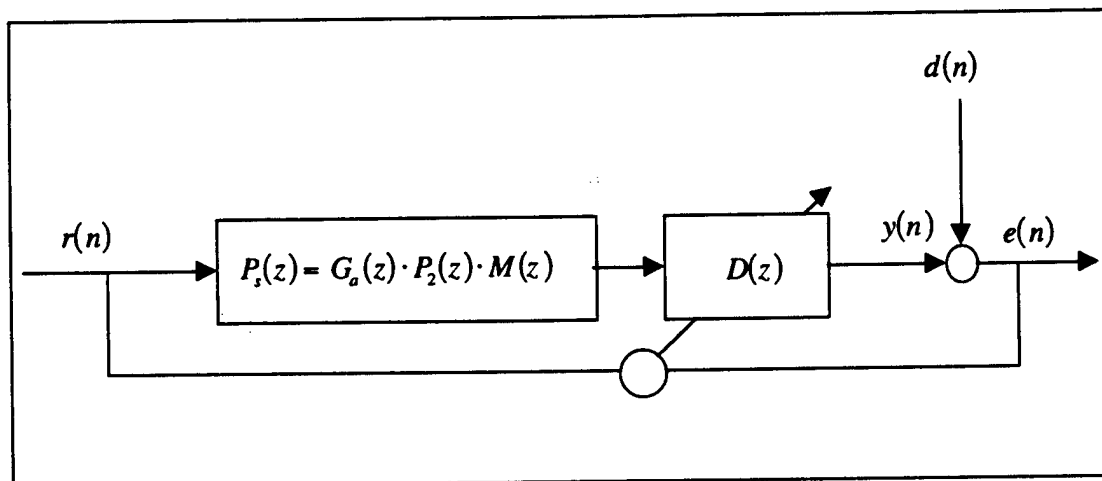


Figure 17 Equivalent system of an actual active noise control system

A solution to the problem of the LMS algorithm with additional transfer functions in the secondary path is provided by the filtered-x algorithm, which was developed independently by both Morgan and Widrow. The filtered-x algorithm takes its name from the fact that the reference signal, commonly denoted $x(n)$, is filtered before being used in the adaptation algorithm. The problem with secondary path transfer functions can be seen more clearly by ignoring the primary sound for a moment. The two inputs to the adaptive algorithm, as shown in figure 15, are a reference signal and an error signal. The LMS algorithm relies on an error signal produced by convolving the reference signal, $x(n)$, with an adaptable filter, $D(z)$. If, instead, there are additional transfer functions between the digital filter and the adaptive algorithm—for instance, the transfer functions of the magnetic bearings and the sensor microphone, shown as $G(s)$ and $M(s)$ in figure 13—the LMS algorithm may become unstable, since the gradient estimate used in the adaptive algorithm is no longer an accurate approximation. The system shown in figure 17, with the secondary path transfer functions moved ahead of the adaptable filter, is mathematically equivalent to figure 13, and would suffer from the same instability. If, however, the secondary path transfer functions could be moved back even further, as shown in figure 18, then the system would return to the original LMS algorithm, with a different input signal. Mathematically, the only difference between this scenario and the system shown in figure 17 is that the input signal to the adaptive algorithm is filtered by the secondary path transfer functions. Therefore, if this filtering is performed with an equivalent model of the secondary path transfer functions, as shown in figure 19, the system will return to an equivalent of the original LMS scenario, and the same adaptive algorithm will work. The filtering of the input signal to the adaptive algorithm by a

model of the secondary path transfer functions is known as the filtered-x LMS algorithm [28]. The filtered-x algorithm does not suffer from the instability problems of the LMS algorithm, as can be seen by comparing figure 20 to figure 16.

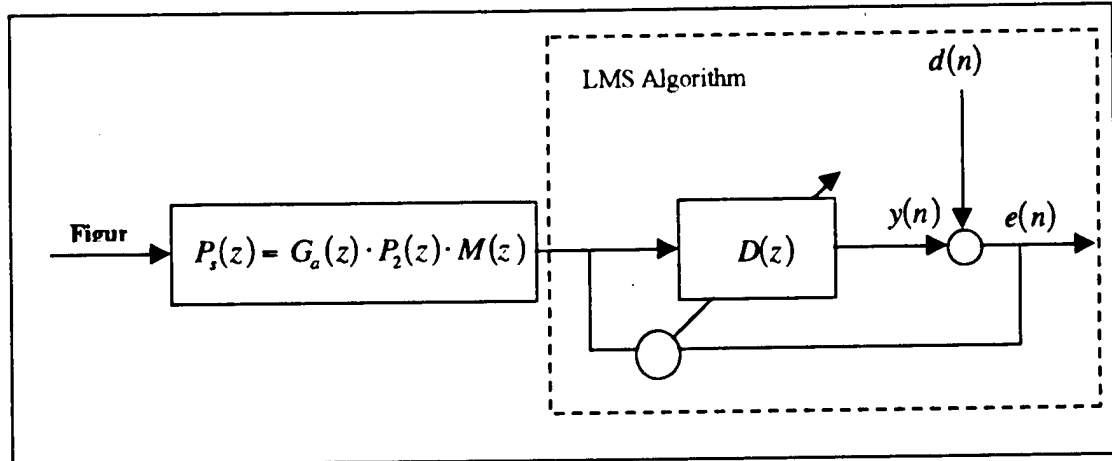


Figure 18 LMS algorithm with a filtered reference signal and no additional secondary path functions

Since the actual secondary path transfer functions are usually physical systems, which cannot be represented exactly by any digital filter, the model, $M(z)$, of the secondary path transfer functions has some error. Even finding a good approximation of the secondary path transfer functions can be difficult, so it is important to know how much error can be present before the inception of instability. The system is quite robust to errors in gain, and can tolerate phase errors up to $\pm 90^\circ$ in the secondary path model [28]. Large phase errors (close to 90°) adversely affect the convergence speed of the algorithm, but errors up to $\pm 30^\circ$ do not have a significant effect on convergence speed.

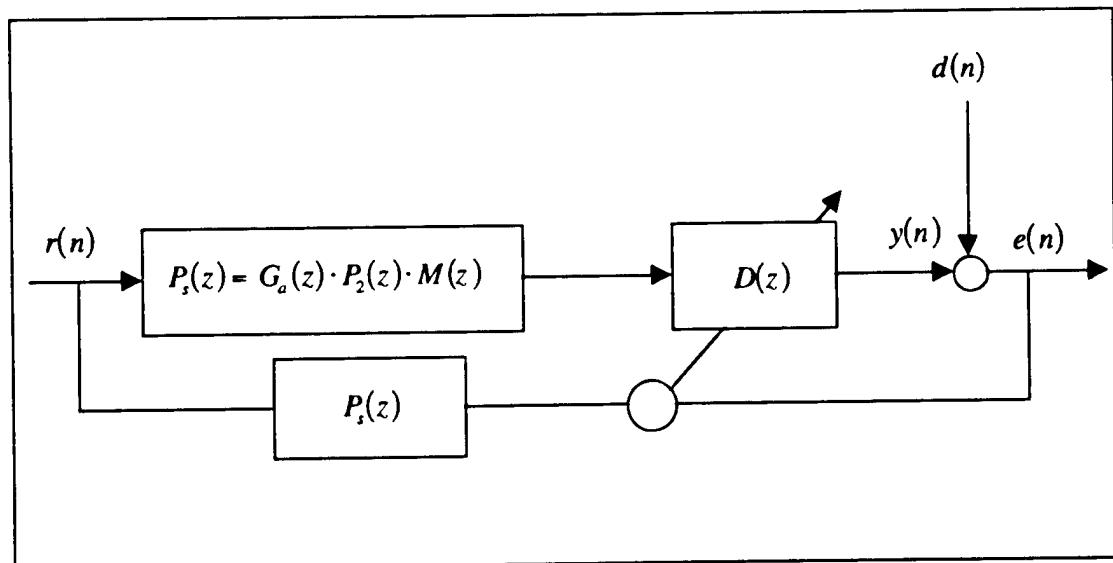


Figure 19 Filtered-x adaptive algorithm

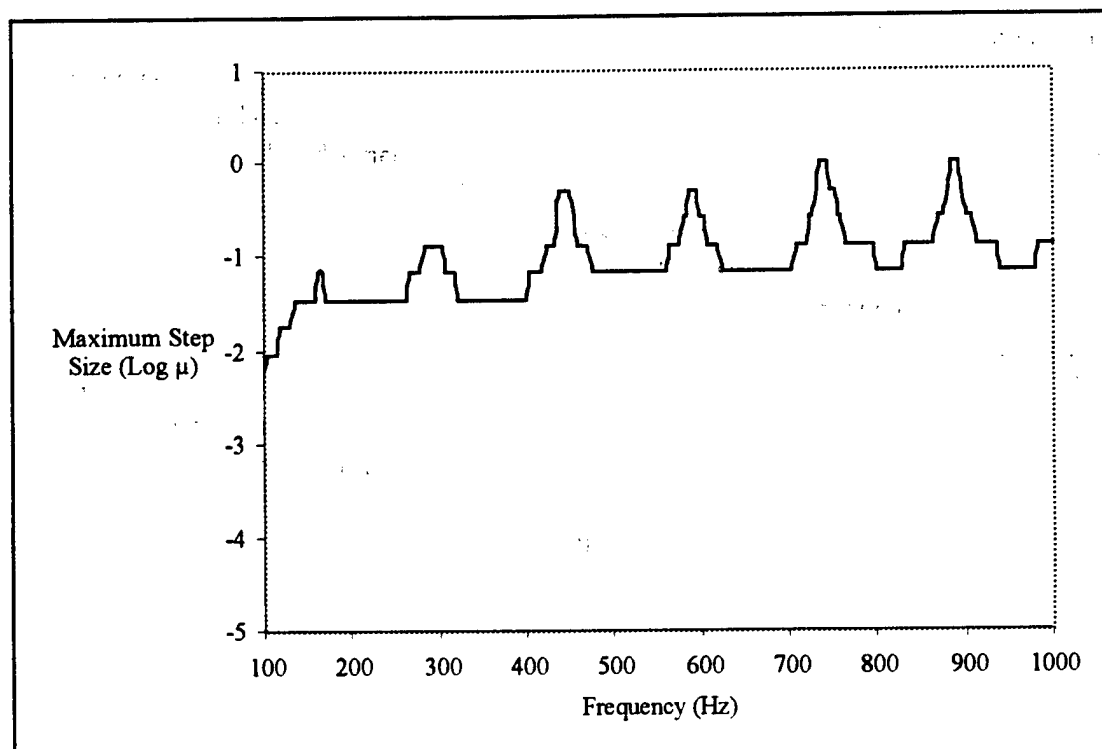


Figure 20 Stability of the filtered-x algorithm

3.2 SIMULATION OF THE NOISE CONTROL SYSTEM

Before testing the actual noise control system, a computer simulation was constructed to evaluate the performance of various noise control algorithms. The path, controller, and actuator transfer functions were all modeled in Matlab, as were the adaptive algorithms. The simulation consists of a Matlab ".m-file," which is a script of Matlab commands run from within the Matlab environment, and various C-coded simulation functions that are called by the .m-file. The .m-file and the filtered-x C-coded simulation files are included in Appendix F. The .m-file establishes system parameters such as sampling time, LMS step size, number of taps, length of the simulation, reference frequencies, transfer functions, and disturbance inputs to the error microphone. All of these values are passed to the filtered-x function and used to produce the final system simulation results. The filtered-x function evaluates all of the transfer functions in the system (as shown in figure 19) at each sample period and stores the output values. The transfer functions used for the two acoustic paths are pure delays, representing the phase shift caused by the travel of the sound waves some distance through air to the error microphone. Two different transfer functions have been used for the magnetic bearings. The first was a discrete-time, or z-domain approximation of the continuous-time theoretical transfer function. The sample intervals for the discrete-time simulation are small enough that the discrete and continuous transfer functions are equivalent within the

frequency ranges of interest in this project. The other transfer function used in the simulation was taken from a curve fit of the bearing frequency response, as measured by a signal analyzer. The comments in the simulation files (Appendix F) provide a more detailed description of the simulation.

Figure 21 shows the time-domain simulation results for a 60 Hz reference frequency with no disturbance sounds. This simulation run used the filtered-x algorithm with a perfect secondary path model. The convergence time is less than 1.2 seconds, and the total noise amplitude asymptotically approaches zero. A less ideal situation was used in the simulation shown in figure 22. This simulation used a pure 100 Hz reference frequency, but included as disturbance sounds the entire 0-200 Hz spectrum taken from actual measurements of the experimental apparatus. Experimental frequency response results were used for the actuator and path transfer functions, while a theoretical transfer function was used for the filtered-x secondary path model. The noise control algorithm was allowed fifteen seconds to converge, and then the results plotted in the frequency domain. A large 100 Hz spike can be seen in the original spectrum (figure 21-a), while the final spectrum shows just the opposite—the 100 Hz tone is by far the quietest in the spectrum. A reduction in amplitude of over 20 dB was achieved in this simulation.

The error microphone used in an active noise control system is subject to a broad range of sounds at many different frequencies, not only at the reference frequency. It is interesting to note the effect that the system has on these other frequencies. The presence of sounds other than the primary and secondary sounds changes the error measurement used by the LMS algorithm, and therefore has some effect on the shape of the secondary sound. However, multiplication of the error signal by a vector of the reference signal inputs acts as a bandpass filter—frequencies close to the reference frequency have a significant effect on the adaptation of the weight vector, while other frequencies have little effect at all. This can be seen in figure 23, which is a plot of the overall gain of the noise control system for a range of disturbance frequencies. The reference frequency in this simulation was again 60 Hz. As can be seen, the amplitude of frequencies close to the reference frequency is altered considerably, but the gain of more distant frequencies is near unity. Disturbance sounds with frequencies in the pass band of the LMS filter appear in the secondary sound, with altered phase and amplitude. For frequencies very close to the reference signal, this phase and amplitude shift causes the passed signal to interfere destructively with the disturbance sound of that frequency, and thus achieves noise reduction at that frequency. However, at frequencies just outside the pass band, the amplitude and phase shift may cause constructive interference, which results in an increase of noise. These effects can be seen in figure 23: frequencies just below the reference frequency have a larger amplitude, while frequencies just above were attenuated. The relative heights of these two peaks can be used to determine the phase error in the filtered-x secondary path model [4].

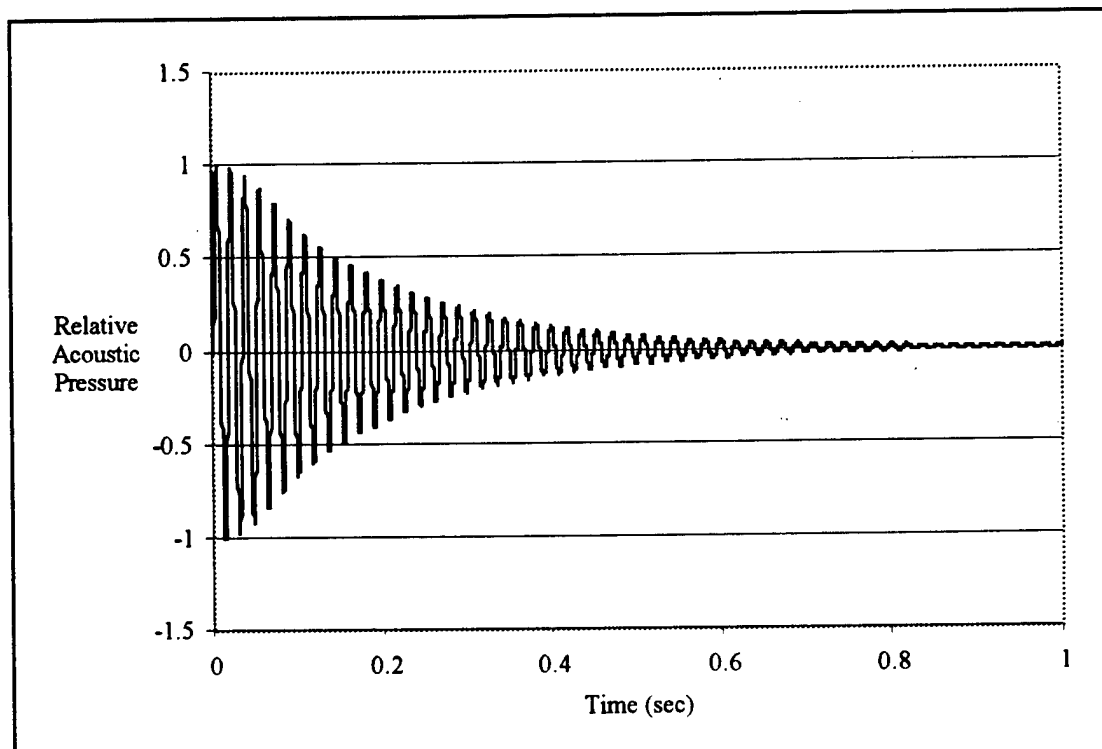


Figure 21 Time-domain results of noise control simulation

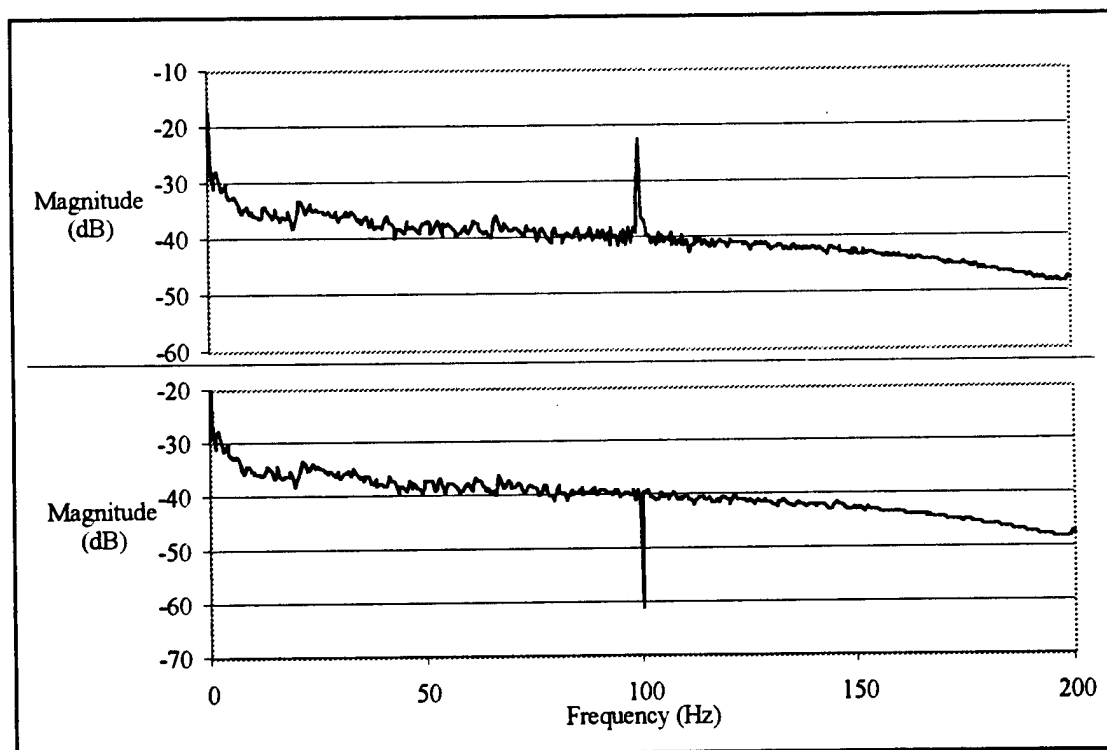


Figure 22 Frequency-domain noise control simulation results—noise control off (top) and on (bottom)

Perhaps the most beneficial use of the simulation constructed for this project is to provide a test for stability. The noise control system was tested with a variety of control algorithms, secondary path transfer functions, and disturbance sounds. After a particular algorithm was tested and tuned so that it provided good noise reduction at a particular frequency, the algorithm was tested over a broad range of reference frequencies. The object of the tests is to determine the largest convergence step size for which the algorithm will converge at a particular frequency. This simulation provided verification of the instability of the LMS algorithm when additional secondary path transfer functions, as was shown in figure 16. The secondary path included a fixed-length delay, so that the phase shift was proportional to frequency. As the phase shift grew past 90° , the first band of instability appeared. At even higher frequencies, where the phase shift passed 270° —which is within 90° of zero phase—the system suddenly became stable again. This phenomenon recurred at even intervals, as was expected from the conditions for instability of the LMS algorithm. Figure 20 shows the results of the stability simulation for the filtered-x algorithm with an exact model of the secondary path—the system is stable for all frequencies.

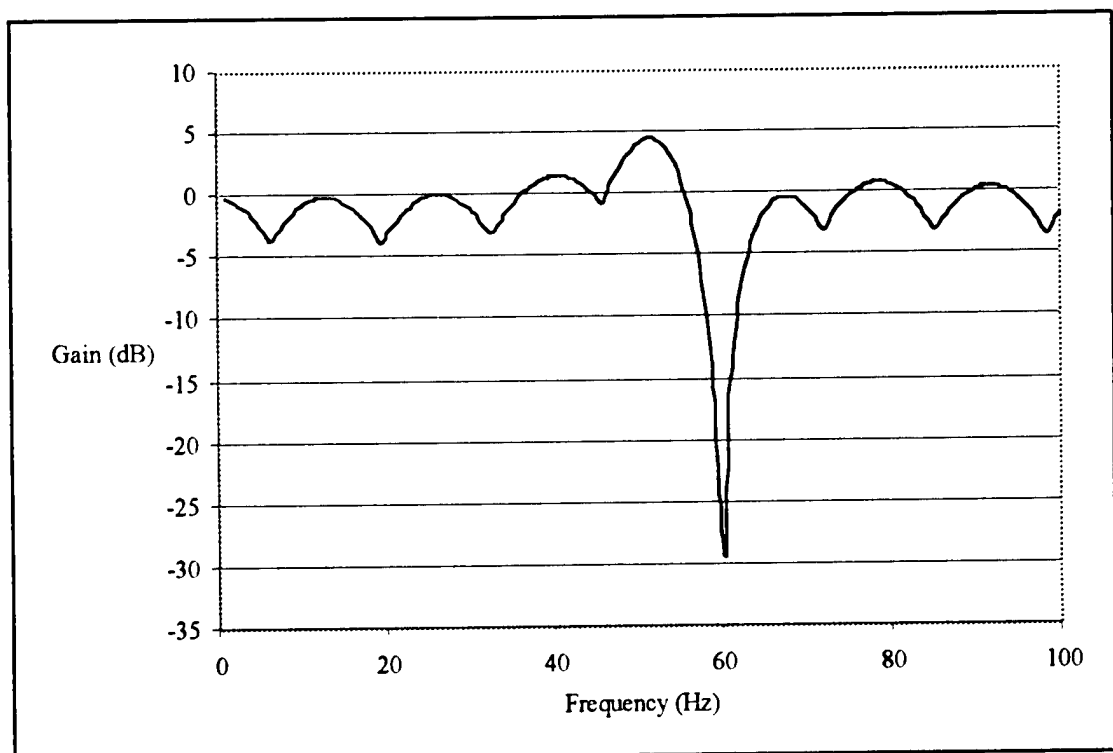


Figure 23 Frequency response of the simulated noise control system

3.3 IMPLEMENTATION

3.3.A Digital Signal Processors

The noise control algorithms for this project are implemented as computer programs, running on dedicated digital signal processors. Digital signal processing requires high-speed computers with specialized capabilities. Digital signal processing boards (DSPs) accept an analog electrical input signal, digitize the signal, manipulate the resulting data through floating point multiplication, division, or other arithmetic processes, and convert the output data to an analog signal⁷. The digital signal processors used in this project are the SHARC ADSP-21061 processors from Analog Devices. Each processor is located on a SHARC EZ-KIT Lite board, which provides the necessary input and output capabilities for the processor. These processors run at a high clock speed—20 MHz—and can perform floating point operations in a single cycle. The specifications of the SHARC are included as Appendix G. Each SHARC DSP card has two input channels and two output channels. This allows simultaneous input of both a feedforward signal and an error signal to one card, and output of either one or two control signals. A C compiler is included with the SHARC processors, so that they can be programmed not only in assembly language, but also in a high-level language.

3.3.B Generation of a Feedforward Signal

The first step in implementation of the filtered-x noise control algorithm is the generation of an appropriate feedforward signal. The frequency of the feedforward signal must be the same as the frequency of the primary sound, so the source of the feedforward signal should be somehow tied to the generation of the primary sound. The objective tones, or primary sounds, in this project are the blade rate and shaft rate of the motor and fan assembly. The shaft rate is a tone that occurs at the same frequency as the rotation of the shaft—a shaft that rotates ten times each second produces a 10 Hz shaft rate tone. The blade rate is a multiple of the shaft rate, produced by the passing of individual blades. A three-bladed fan rotating at 10 Hz would have a 30 Hz blade rate. As both blade rate and shaft rate are related to the speed of shaft rotation, a tachometer measuring shaft rotation would provide an appropriate reference signal. The tachometer used in this project is a position sensor seated next to a notched wheel on the shaft. Each time the notch passes the position sensor, the sensor output voltages spikes, so that a train of spikes at the shaft rate is generated. A DSP is used to measure the frequency of these spikes and to generate a sine wave of the same frequency, which is the desired form of

⁷ In a digital signal processor, the input and output of data is done at a regular interval, known as the sampling time or sample period. The sampling frequency, or sample rate, is the inverse of the sample period.

the reference signal. A lookup-table algorithm⁸ is used, so that the DSP can easily multiply the shaft frequency in order to generate another sine wave at the blade rate. Listings of the program code for generation of the feedforward signal are included in Appendix I.

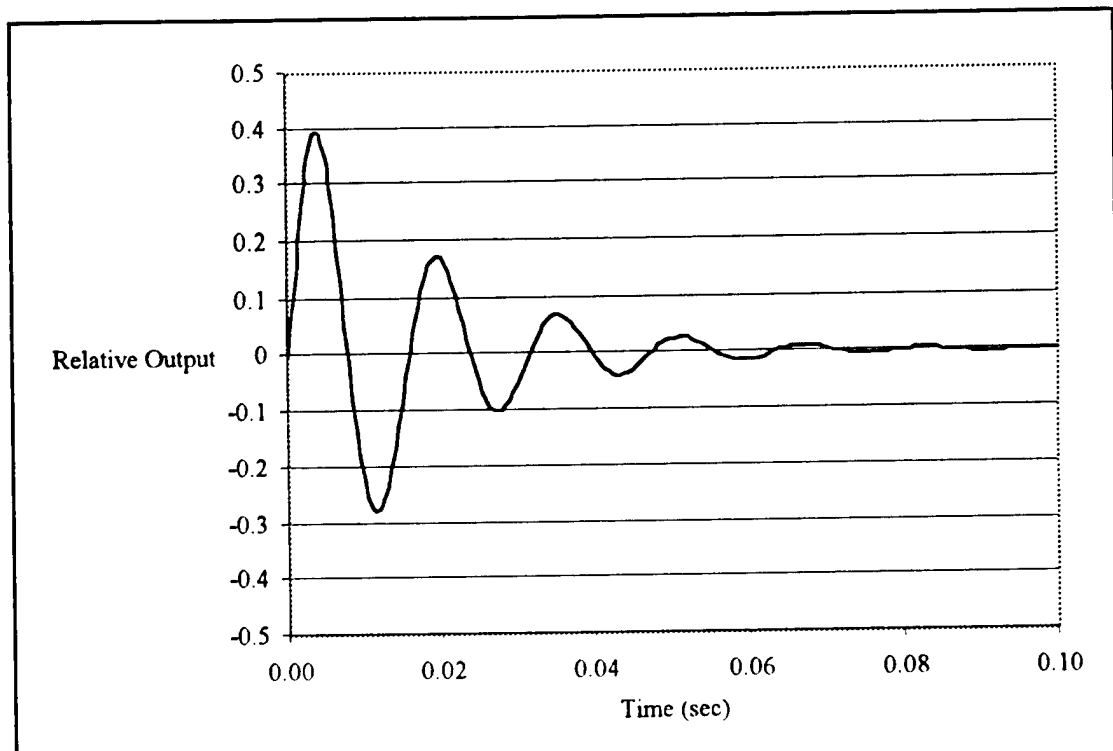


Figure 24 Closed-loop magnetic bearing system impulse response

3.3.C Control Algorithm

A filtered-x algorithm was chosen as the control algorithm for this project, and much of the program code follows directly from the mathematical functions that constitute the filtered-x method. The important factors in the control algorithm that must be selected are the sampling time, length of the weight vector, convergence step size, and secondary path model. The sampling time is limited by the speed of the processor and the number of operations that must be performed—if the sample rate is too fast, the DSP will not be able to perform all of the necessary operations in time to provide the next output value. On the other hand, if the sampling frequency is too slow, the output signal quality will become poor and may have an adverse effect on the performance of the system. For the DSP boards used in this project, the minimum sample rate at which the processor could generate a signal of appropriate quality was found to be 8 kHz. Once the sampling time is set, the length of the weight vector follows from measurement of the

⁸ The lookup table algorithm is explained in the comments in the program listing.

actuator impulse response. As discussed in section 3.1, the length of the impulse response of the filter should match the length of the impulse response of the actuator. The length of the impulse response of the filter is the product of the length of the weight vector and the sampling time. The impulse response of the actuator was determined to be 0.1 seconds by simulation, as shown in figure 24. With an 8 kHz sample rate, the filter would need eight hundred taps in its weight vector to match the length of the actuator impulse response. Unfortunately, the processor is not fast enough to handle eight hundred taps at 8 kHz; the number of taps was reduced to 100, which is near the limit of the capabilities of the processor. The reduced number of weights may have degraded the performance of the noise control system, but did not prevent the system from achieving good results. A listing of the program used for final experimentation is included as Appendix H.

3.3.D Error Signal Feedback

A system with feedback has several advantages over open-loop control systems, but also has the added difficulty of obtaining a useful feedback signal. In the case of a noise control system, the feedback is provided by a microphone. The number, quality, and placement of sensors all affect the overall performance of the system. In systems with multiple actuators, it is best to have multiple sensors. Multiple sensors can also be used in a system with a single actuator, although only one is used for this project. The microphone, which was provided by the Annapolis detachment of the Carderock Naval Surface Warfare Center, contains its own signal conditioning circuitry, which provides high-quality output, suitable for use without further filtering. The sensor should be placed in a region that is "acoustically coupled" to the actuator—that is, the output of the actuator should have a significant effect on the measurements of the sensor. If the sensor is placed in a location that is not well coupled to the actuator, the noise control system will be unable to affect the total noise level as measured by the microphone. In such a situation, the controller will continue to increase the amplitude of the secondary sound, seeking to produce some change in the measured error signal. Ultimately, the secondary sound will either increase until some element of the noise control system saturates, or will reduce the noise level near the error microphone, at the cost of increased noise levels elsewhere. In this project, the sensor was placed in front of the fan blade, between six and twelve inches away, where the output of the magnetic bearing sound actuator is the strongest. The test results of the physical system are discussed later.

4 ACOUSTICS

The underlying acoustical principles of active sound control are significantly more complicated than the simple addition of two out of phase sine waves. The actual acoustic fields involved in this project are three-dimensional fields produced from several nearby sources, rather than one dimensional waves from a point source. A brief discussion on noise sources, radiation patterns, and mechanisms of sound control is provided here.

4.1 PRIMARY NOISE SOURCES

Sound is a periodic variation in pressure in a medium, such as air or water. For volumes not painful to the human ear, this pressure variation is extremely small. The sound produced from normal operation of the apparatus arises primarily from three sources: steady-loading noise, nonuniform flow fields, and random flow fields [14]. Random flow field noises result from the passing of fan blades through the unsteady, randomly occurring flow fields that occur in a real physical environment. The forces that occur between these flow fields and the fan blade yield a broadband noise, usually several decibels below the level of the blade rate. The other two noise sources—steady loading noise and nonuniform flow field noise—occur primarily at the blade rate. Steady loading noise is a product of the steady thrust forces produced by each fan blade. As the fan blades move relative to a stationary observer, the forces on the fan blades also move in a periodic manner, thus generating a tone at the blade rate. The steady loading noise is greatest to the sides of the fan, where the relative motion of the fan blades to a stationary observer is greatest. The nonuniform flow field noise is produced from the movement of fan blades through different steady flow fields along the path of the blades. Any obstruction to the flow of air through the fan generates a constant perturbation in the overall flow field, and the passing of the blades through these perturbations generates a tone at the blade rate. From experimentation, the radiation pattern of the apparatus was found to be stronger to the sides, as expected.

4.2 SECONDARY NOISE SOURCE

The secondary sound is produced by the vibration of the fan blade by the magnetic bearings. As the magnetic bearings change the position of the shaft, the shaft and fan blade move with some velocity. The movement of the broad face of the fan blade in the lateral direction generates a pressure wave in the air proportional to the velocity at which the blade moves. Since the position, and therefore velocity of the shaft

and fan blade are altering periodically, the pressure waves will have a periodic variation. The frequency of this variation is the frequency of the secondary sound.

While the natural radiation pattern of the apparatus is constant, the field of the actuator can be varied. The shaft and fan blade are supported in five degrees of freedom by the magnetic bearings, offering a wide variety of possible directions and patterns in which to vibrate the fan blade. For this project, only the lateral (thrust) direction was used, as vibration in this direction will theoretically produce the greatest amplitude secondary sound. Future research will consider the optimal directions for vibration to effect the greatest noise reduction possible. Since the fan is vibrated laterally by the actuator, the actuator radiation pattern does not match that of the normal noise from the apparatus. While the natural noise is slightly louder to the sides, the actuator field has a greater amplitude to the front and rear of the fan. This difference in radiation patterns does not mean that global noise reduction cannot be achieved, but that greater noise reduction will be achieved along the strong axes of the actuator field (the lateral axis of the fan blade and shaft).

4.3 MECHANISM OF SOUND CONTROL

The interactions of varying pressure fields in the air can be extremely complicated. However, acoustic pressures are so small that their behavior can be accurately described by superposition. That is, for small amplitudes, multiple pressure waves in the air combine additively [17]. While the effects of this superposition are easily demonstrated by simple addition of two sine waves, as in figure 1, the physical mechanism of sound control begs a more thorough explanation.

As with electrical or magnetic signals, acoustic signals are described by two variables. For sound, the across variable is pressure and the through variable is volume flow rate, or volume velocity. The total power in an acoustic signal is derived from the product of these two variables. Since pressure and volume velocity are complex numbers, with oscillating values, an average of their magnitudes is used to describe the power in the signal. The following derivation is from [17]. The formula for the root-mean-square, or RMS power in an acoustic signal is:

$$W = \frac{1}{2} \operatorname{Re}(q^* \cdot p) \quad (4.1)$$

W : Acoustic power output of a source

$\operatorname{Re}(x)$: Real part of x

q : Complex volume velocity of the source

p : Complex acoustic pressure at the source

The acoustic pressure at a source is also related to the volume velocity of that source by the "acoustic impedance," which is equivalent to resistance in a electric circuit⁹:

$$p_p = Z_{pp} \cdot q_p + Z_{ps} \cdot q_s \quad (4.2)$$

$$p_s = Z_{sp} \cdot q_p + Z_{ss} \cdot q_s \quad (4.3)$$

Z_{pp} : Acoustic "input" impedance seen by the primary source

$Z_{sp} = Z_{ps}$: Acoustic "transfer" impedance between the two sources

Z_{ss} : Acoustic "input" impedance seen by the secondary source

The total output of the two sources taken together, in terms of volume velocity, is:

$$W_T = W_p + W_s \quad (4.4)$$

$$W_T = \frac{1}{2} \left[|q_s|^2 \cdot R_{ss} + q_s^* \cdot R_{sp} \cdot q_p + q_p^* \cdot R_{sp} \cdot q_s + |q_p|^2 \cdot R_{pp} \right] \quad (4.5)$$

R_{ss} : Real part of Z_{ss}

R_{sp} : Real part of Z_{sp}

R_{pp} : Real part of Z_{pp}

The two terms involving input impedance account for the power output of each source acting alone, and will always be positive. The remaining two terms (cross terms) account for the effects of the two sources on one another. In order for noise reduction to be achieved, the product of the volume velocities of the two sources and the transfer impedance must be negative:

$$q_s^* \cdot R_{sp} \cdot q_p < 0 \quad (4.6)$$

The transfer impedance is given as a function of the distance between the two sources in [17]:

$$R_{sp} = \frac{\omega^2 \cdot \rho}{4\pi \cdot c_0} \operatorname{sinc}\left(\frac{2\pi \cdot r}{\lambda}\right) \quad (4.7)$$

ω : Frequency of the sound

ρ : Density of the medium

c_0 : Speed of sound

r : Distance between the primary and secondary sources

λ : Wavelength of the sound

⁹ The subscript "p" is used to denote properties of the primary source, while "s" is used for the secondary

The impedance is positive when the distance between the primary and secondary sources is less than half of the wavelength of the primary sound, and negative when the separation is between $.5\lambda$ and λ . For greater separations than this, the impedance will alternate between negative and positive values, but will be too small to achieve significant global noise reduction. Perhaps the greatest advantage of a noise control system using magnetic bearings is that the fan blade, which is the source for much of the primary sound, is also used as the secondary sound source. As shown in the above equations, significant global noise reductions are only possible when the primary and secondary sources are close to one another. Figure 25 shows a comparison of two sources close together and two sources far apart. The concentric circles in the figures represent the peak amplitude of the sound waves traveling outward from the sources. Wherever two circles intersect, perfectly constructive interference occurs, doubling the sound amplitude at that point. Wherever the peak of a wave from one source falls precisely between two peaks from the other source, perfectly destructive interference occurs, eliminating sound from either source at that point.

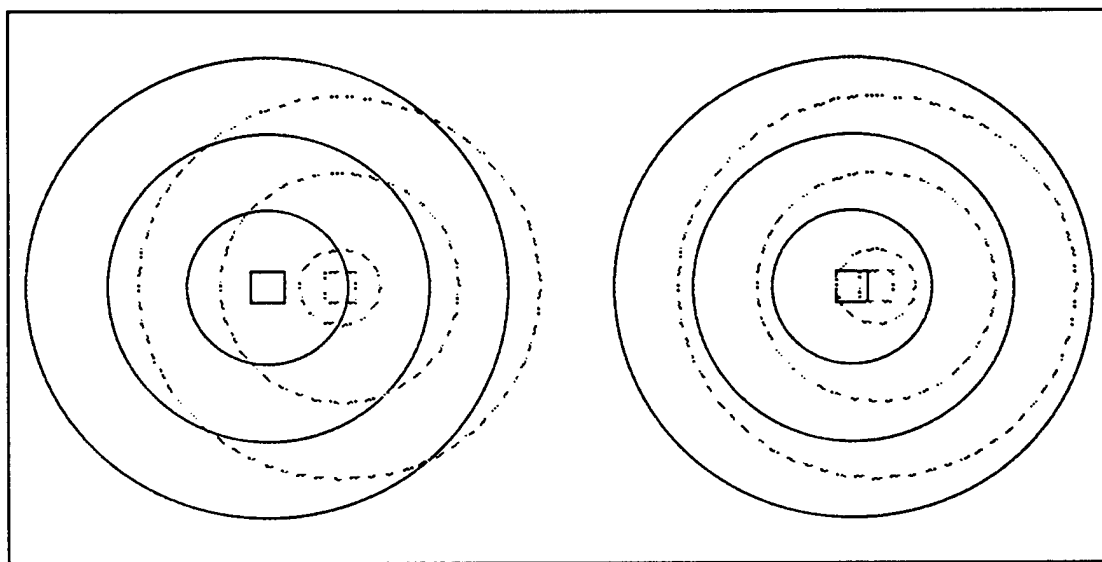


Figure 25 Global effects from two distant sources (left) and two close sources (right)

In order to satisfy equation (4.6), the volume velocities of the two sources must be out of phase when the sources are less than one half wavelength apart, and in phase when they are more than one half wavelength apart. The optimal secondary source volume velocity is shown in [17] to be¹⁰:

$$q_s = -q_p \text{sinc}(k \cdot r) \quad (4.8)$$

source.

¹⁰ This equation applies for monopole sources acting in a free field.

The ideal secondary sound is thus either exactly in phase or exactly out of phase with the primary source, and has decreasing amplitude as the sources grow more distant from one another.

The fundamental concept of active noise control often seems to violate the principle of conservation of energy: sound, or energy, from one source added to sound from another source yields less sound than that from the original source. The secondary source does not add acoustic power to the environment. In fact, it has been shown in [17] that the acoustic power output of the secondary source is zero when optimally adjusted. Rather, the secondary source serves to reduce the impedance seen by the primary source, by reducing the pressure at the primary source when the volume velocity of the source is at a maximum [5].

5 RESULTS

The objectives of this project focused on the ability to produce and control sound at particular frequencies using magnetic bearings. Therefore, the vast majority of the data collected are in the form of spectral power density measurements. Spectral power density is the amount of acoustic power, or sound, measured by a microphone and broken down by frequency, so that the amplitude of any particular frequency can be seen. In looking at a graph of spectral power density, the frequencies appear on the horizontal axis, and the amplitude appears on the vertical axis. For all of the data in this report, frequency is given in Hertz (Hz) and amplitude is given in decibels (dB). A difference of three decibels between two power measurements indicates that the higher measurement has twice as much power as the lower measurement. A reduction in sound level of three decibels thus corresponds to halving the volume of a tone.

Data were gathered for this project in three primary areas. The first concerns the ability of the magnetic bearings to produce a sound. The second demonstrates the ability of a sound control system using magnetic bearings to reduce the sound level at the error microphone. The third tests the effects of the sound control system at other points in space when the sound at the error microphone has been attenuated.

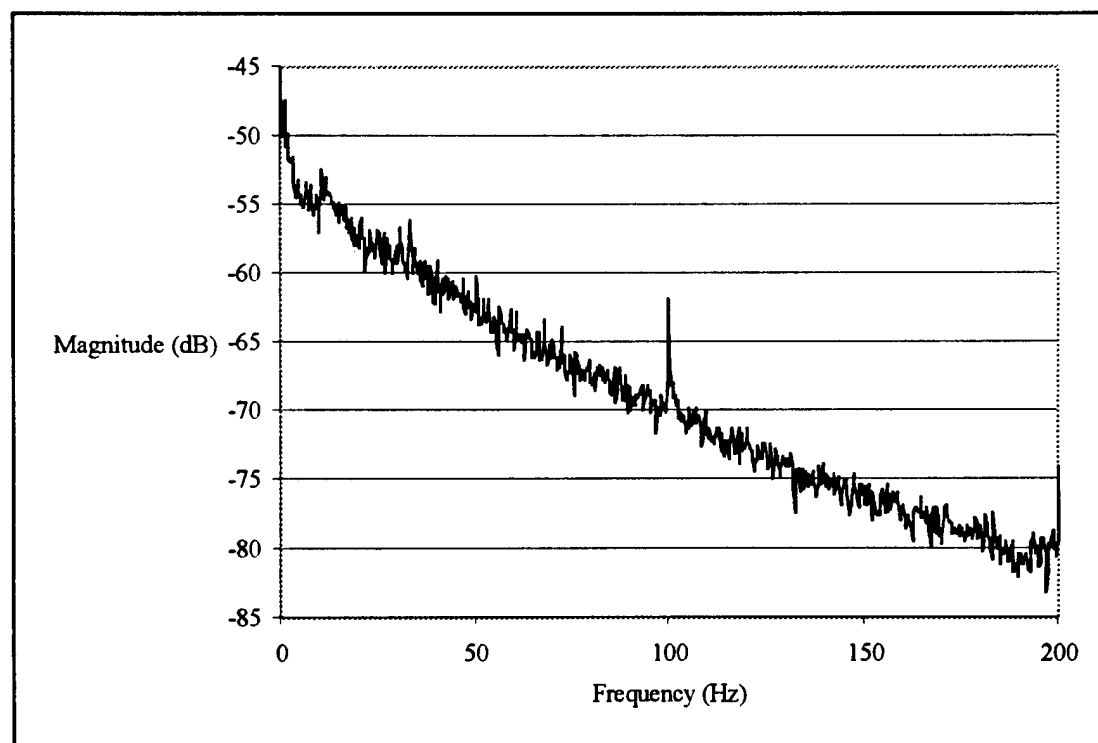


Figure 26 Power spectral density with apparatus operating at 2000 rpm

5.1 ACTUATOR TESTS

The first objective of this project was to demonstrate that magnetic bearings can be used as an actuator for active noise control. In order to be used as an actuator, the magnetic bearings must be able to produce a narrowband tone of amplitude comparable to the nominal amplitude of the primary sound. The first measurements taken in this series of experiments were used to determine the amplitude of the motor/fan blade apparatus running at a constant speed, and to find a rough radiation pattern. After the normal, uncontrolled sound levels were measured, a pure tone was input to the magnetic bearing controller, and the acoustic levels were measured again. From a comparison of these two series of measurements, the differences in amplitude and radiation patterns of the natural noise of the apparatus and the intentional noise from the magnetic bearing actuators was determined.

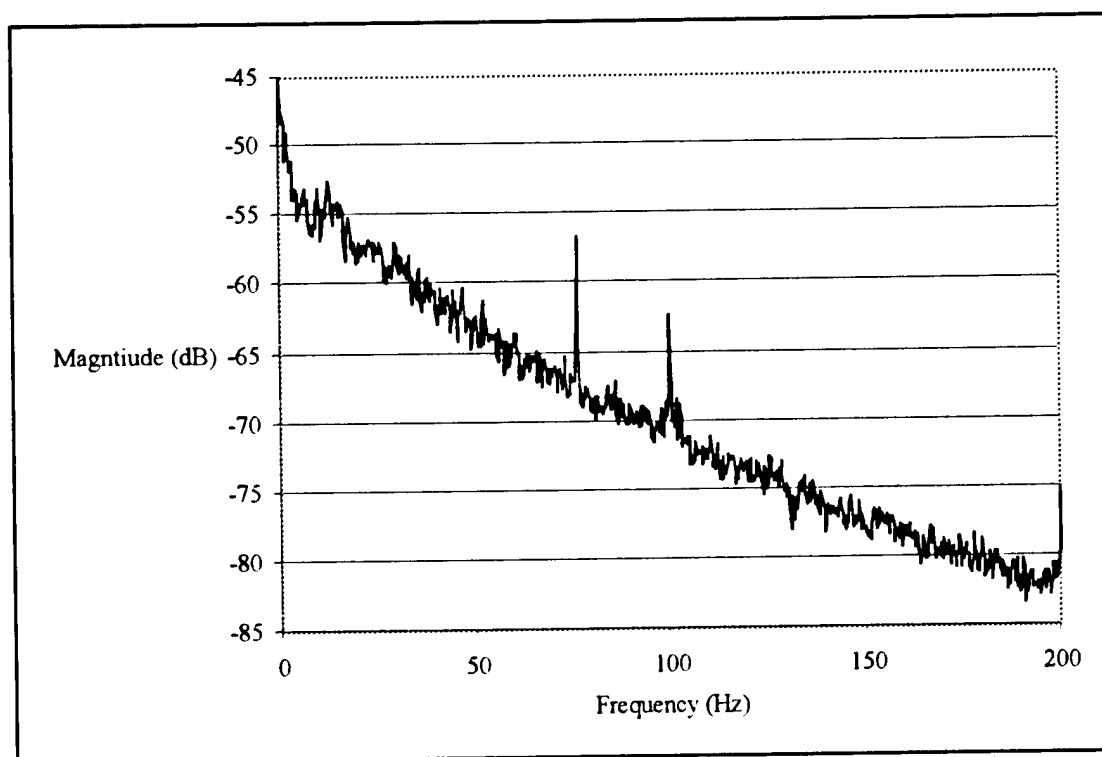


Figure 27 Power spectrum with apparatus operating at 2000 rpm and a 76.5 Hz input to the magnetic bearing controller

Figure 26 shows a baseline noise measurement taken with the apparatus running at 2000 rpm. The measurement was taken six inches in front of the center of the fan blade. At this location, the spectrum is dominated by a single spike at 100 Hz, corresponding to the blade rate. Next, a series of pure-tone signals were used to drive the magnetic bearings, in order to test their ability as an actuator for producing sound. These tone inputs were added to the reference input of the thrust bearing only, so that the shaft and fan blade were vibrating in the lateral direction only. The results of two tests are

shown in figures 27 and 28. The microphone for these measurements was in the same location as for the measurement in figure 26, and the apparatus was running at the same speed. Figure 27 shows two spikes in the power spectrum: the 100 Hz spike corresponding to the blade rate is still present, but a larger spike at 76.5 Hz—the tone input to the thrust bearing controller—is also visible. For the measurement shown in figure 28, the input tone was shifted to 104 Hz. At this frequency, the height of the spike from the intentional tone is approximately the same as that of the blade rate tone. It is clear from these measurements that the magnetic bearings are capable of producing a narrowband tone of sufficient amplitude for sound control. Comparison of the sound levels to the sides of the fan blade showed the 100 Hz spike from the blade rate to be approximately 2 dB greater than the level of the sound generated by the magnetic bearings. As discussed in section 4.2, this difference in radiation patterns does not prevent global noise control from being achieved.

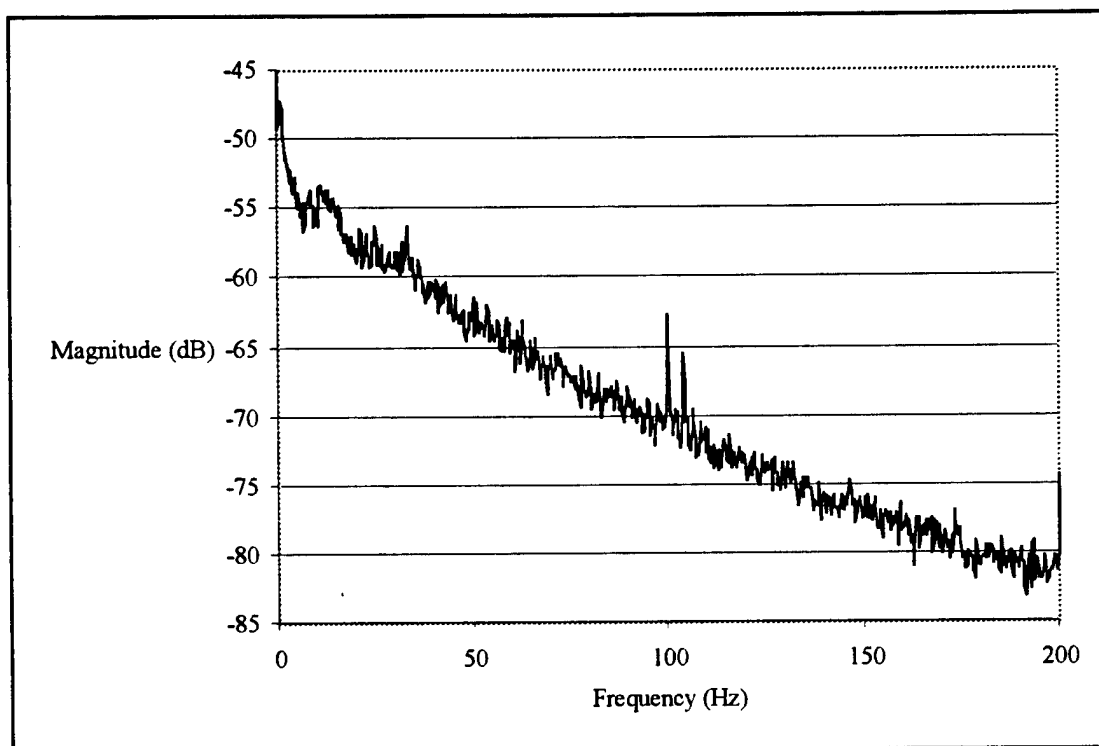


Figure 28 Power spectrum with apparatus operating at 2000 rpm and a 104 Hz input to the magnetic bearing controller

5.2 NOISE CONTROL SYSTEM TESTS

Having demonstrated that the magnetic bearing actuators can generate a narrowband tone of sufficient magnitude for use in an active sound control scheme, the next task is to employ the actuators in an actual noise control system. These tests use the

same implementations described earlier in this paper with respect to hardware and software. The microphone was placed ten inches in front of the fan blade and three inches to the right. The blade rate and shaft rate tones there are both prominent in the acoustic signature. The sound control algorithm was run and allowed approximately one minute to converge before measurements were taken.

The trial shown in figure 31 demonstrates a 4 dB reduction in the blade rate tone. Figures 29 and 30 show the power spectra with noise control off and on, respectively. The shape and height of the 100 Hz peak are clearly different with noise control on. A magnified view of the frequencies around the blade rate is shown in figure 32. The single prominent spike at 100 Hz was reduced to a broad hump, showing that the acoustic signature of the plant can be reduced and altered significantly. A second trial, with the error microphone placed farther from the fan blade, is shown in figures 33-35. In this trial, the 100 Hz blade rate spike is entirely eliminated from the power spectrum, although the reduction in amplitude is still approximately 4 dB.

Since the noise control scheme in these tests involves the filtered-x algorithm, the primary factor in determining the stability of the algorithm is the phase shift provided by the secondary path model. The algorithm is stable, provided that the model is accurate to within 90° of phase. For best results, the model should be accurate to within 30° . In order to test these theoretical results, the noise control system was run for secondary path models providing between 0° and 345° of phase shift, in 15° increments. Test results confirm that the algorithm behaves best within a 60° range of phase for the secondary path model.

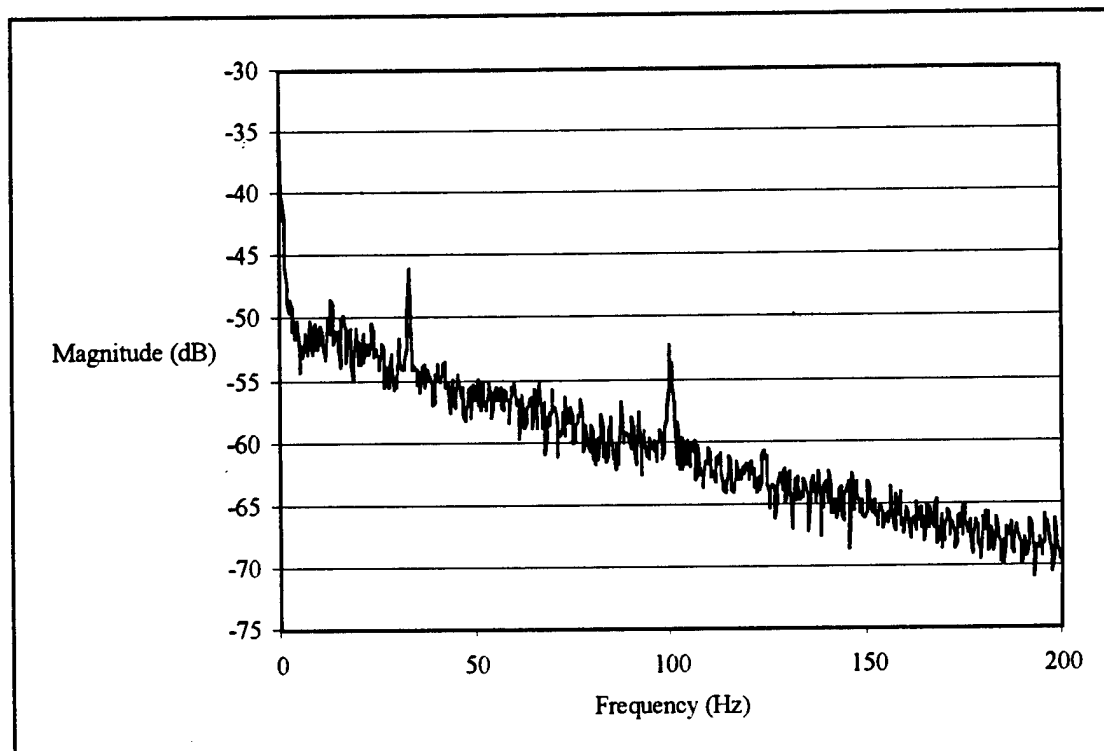


Figure 29 Power spectrum with noise control off (trial 1)

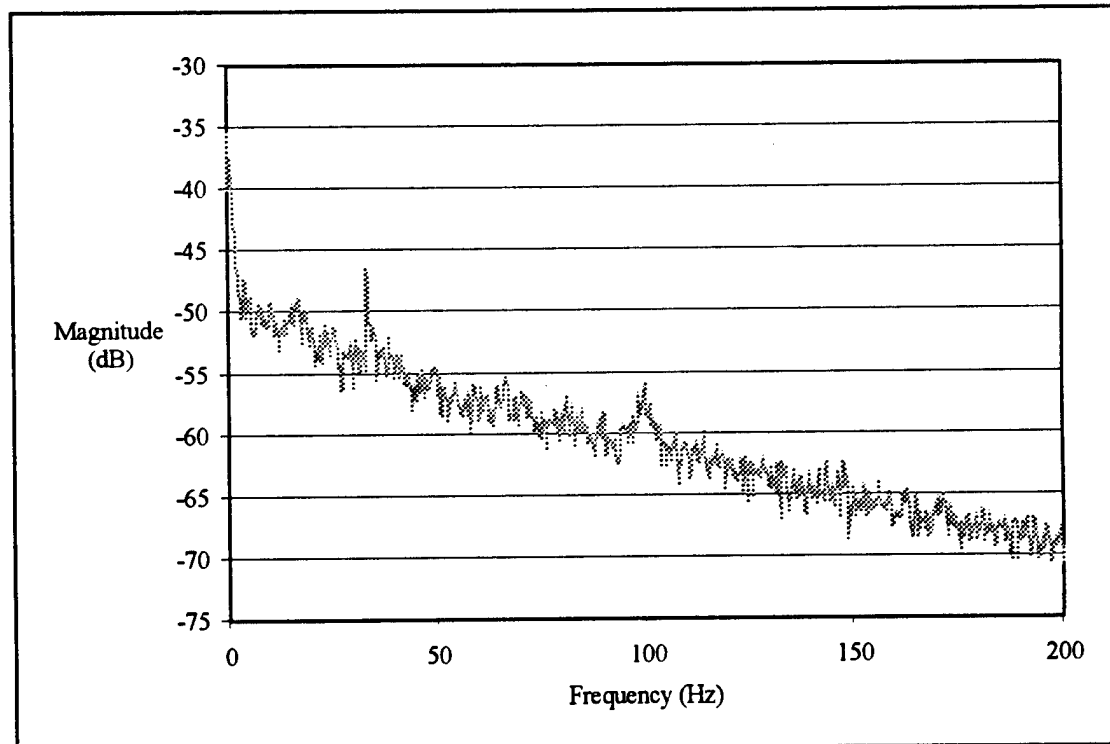


Figure 30 Power spectrum with noise control on (trial 1)

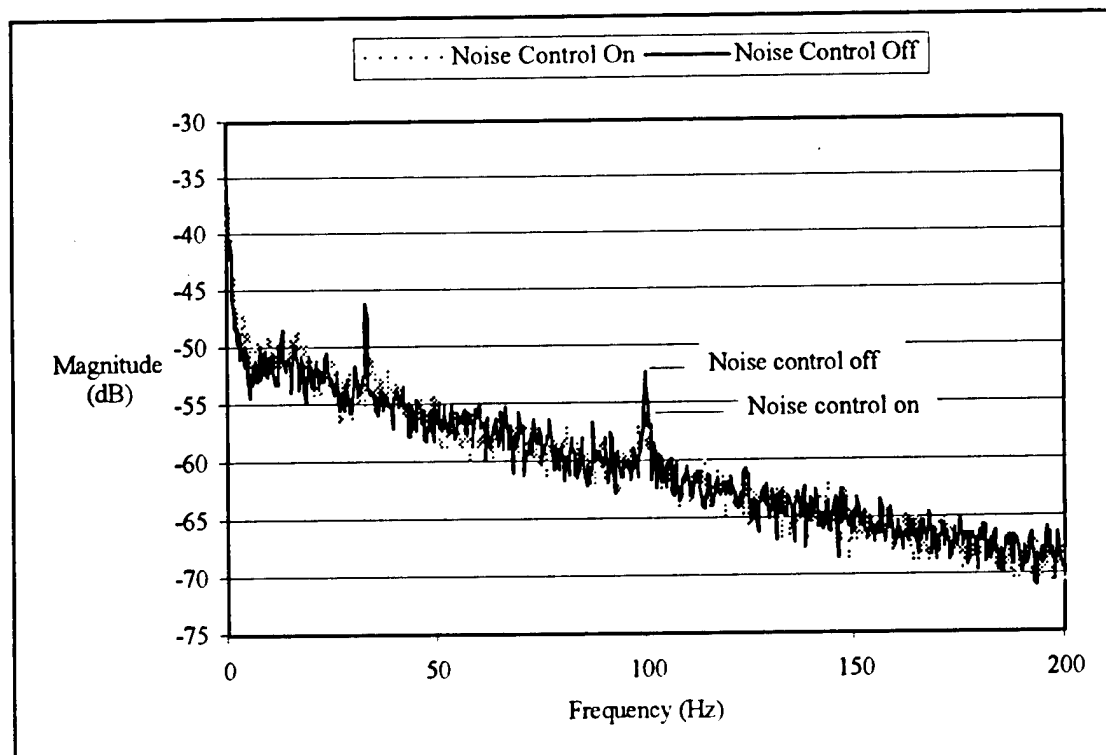


Figure 31 Comparison of power spectra with noise control off and on (trial 1)

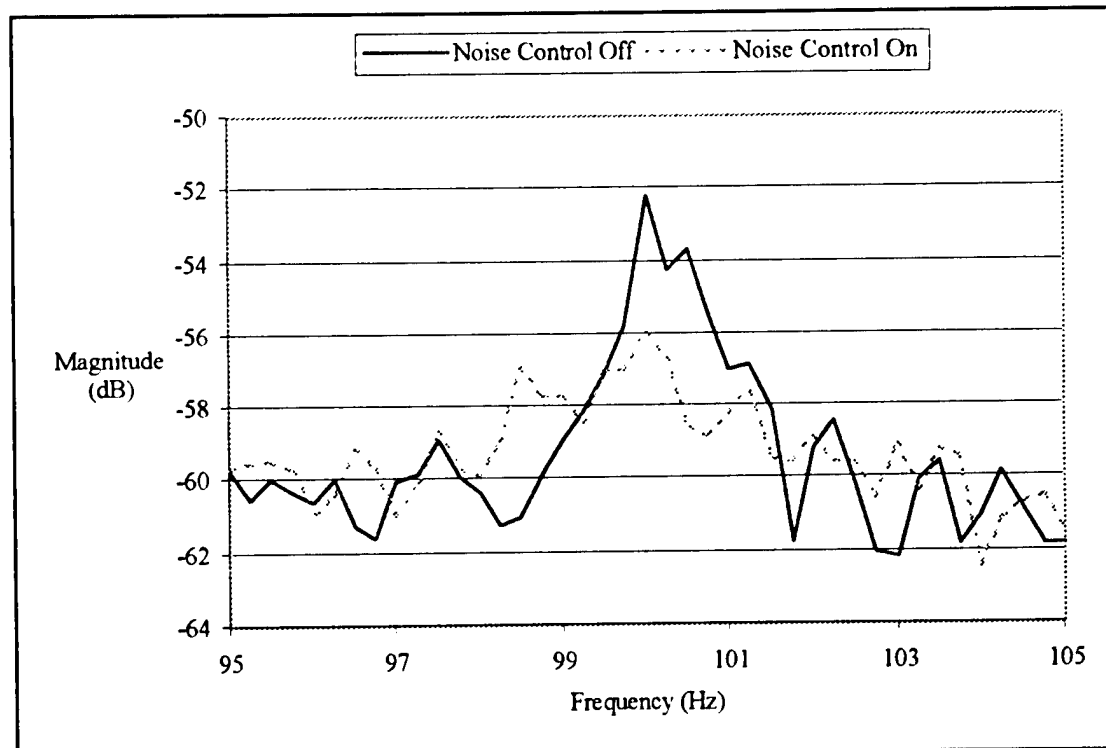


Figure 32 Comparison of power spectra around the 100 Hz blade rate spike (trial 1)

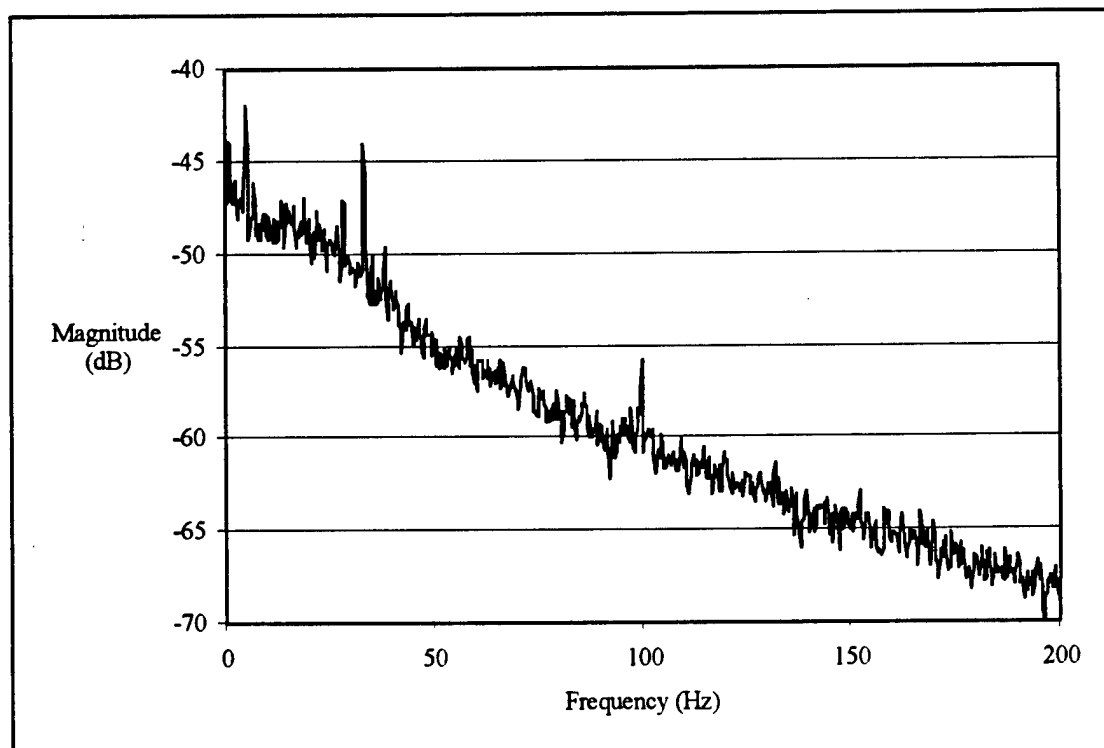


Figure 33 Power spectrum with noise control off (trial 2)

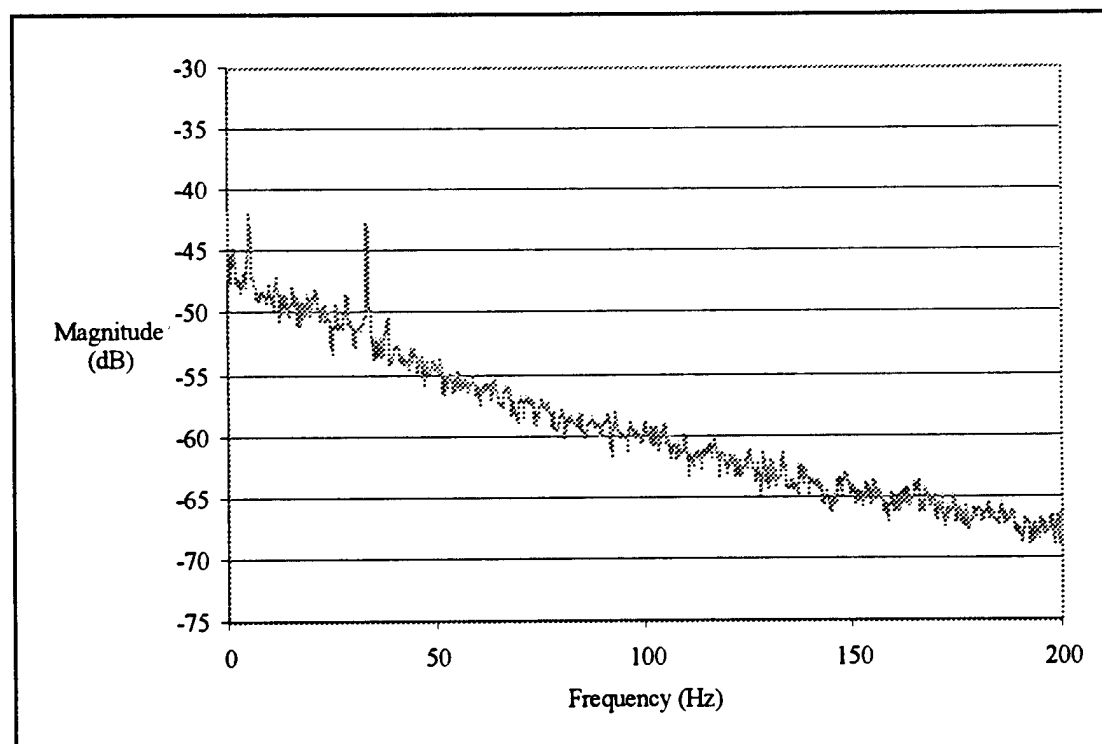


Figure 34 Power spectrum with noise control on (trial 2)

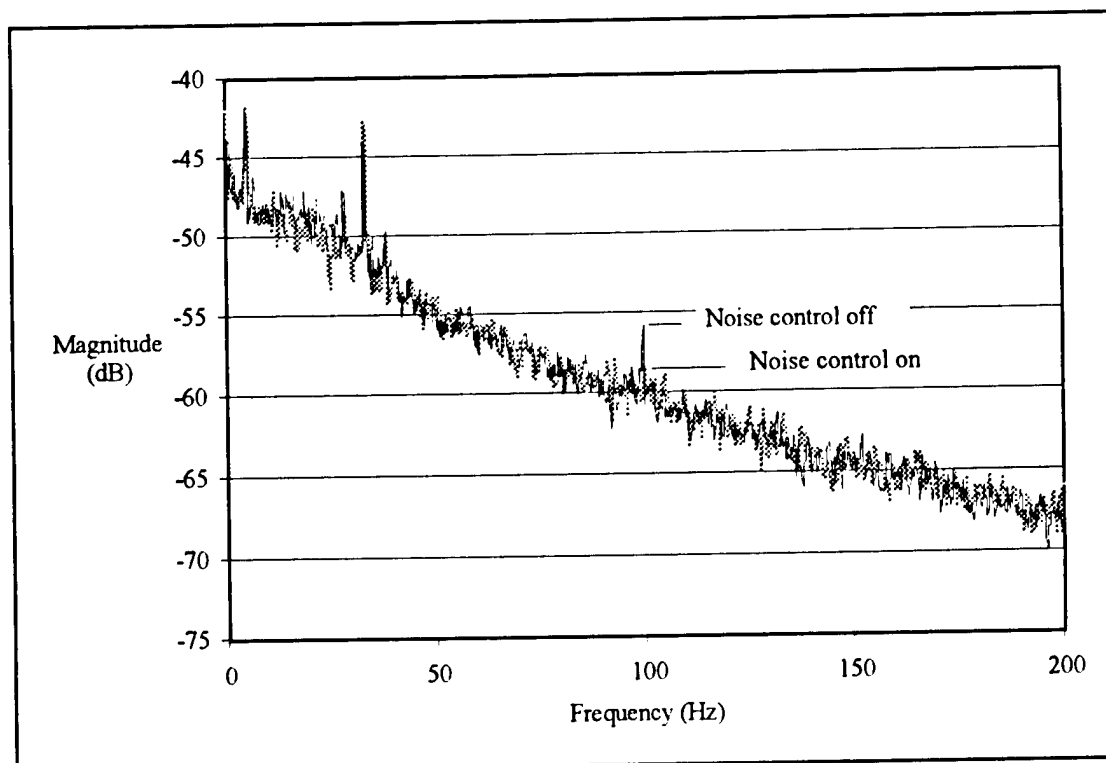


Figure 35 Comparison of power spectra with noise control off and on (trial 2)

5.3 GLOBAL EFFECTS TESTS

The tests described above prove that noise control can be achieved at or near the error microphone. The primary theoretical advantage of the magnetic bearing actuator system is its ability to achieve good global noise control. In order to test this, several measurements were taken at locations other than that of the error microphone. While the noise reduction at areas away from the error microphone was not as great as at the error microphone, a reduction of 3 dB was seen six feet away from the error microphone.

6 SUMMARY

This project has demonstrated that magnetic bearings can be used effectively as an actuator for noise control. The magnetic bearing system was shown to have several advantages over conventional noise control techniques. Effective global sound control from a single sensor and actuator was demonstrated.

In the course of this project, several achievements were made. A complete magnetic bearing system was designed and assembled. The bearings were tested successfully at speeds in excess of 3000 rpm. A detailed computer simulation of a noise control system using the magnetic bearings was constructed, and the results used in the design of an actual noise control system. The actual noise control system was implemented and tested, with noise reductions of approximately 4 dB achieved.

Further research in this project will include refining the noise control algorithm and testing different vibration patterns and directions using the magnetic bearings. Further testing will be conducted to better quantify the degree of global noise control that has been achieved. Additionally, it would be of some interest to analyze the effects of fan blade vibration on the structure-borne noise in the system and on the efficiency of the fan blade.

APPENDIX A—LINEAR APPROXIMATION PROGRAM

```

% findk.m
%
% An m-file to find force-current and force-position constants for the AMBER
% project radial bearings. This file first calculates points along the nonlinear curves,
% fi and fs. The program then finds the slope between the two points located on
% either side of the operating-point values for force and current. These slopes serve
% as the linear approximations of the force-current and force-position relationships
% about the magnetic bearing operating point.
%
% by John Wiggins
% 25SEP97
% Adapted from a similar file by Prof. George Piper

muo = 1.2566637E-6; % Permeability of free space(N/A^2)
mur = 700; % Relative permeability of bearing material
Aa = 6.83E-5; % Magnet cross-sectional area (m^2)
n = 140; % Turns of wire per pole
l = .02565; % Average length of magnetic circuit (m)
theta = 22.5 / 57.3 % Angle of poles
s0 = 3.81E-4; % Operating-point air gap (m)
it0 = 1.25; % Operating-point current for top bearing (A)
ib0 = 0.25 % Operating-point current for bottom bearing (A)

i = linspace( 0, 3, 100 ); % current values (independent variable)
s = linspace( 0.0003, 0.0004, 100 ); % position values (independent variable)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FORCE-CURRENT CALCULATIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The position value is held constant at the operating point value, and the force of
% the bearing is calculated for each of the current values in the array "i"
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Force of magnetic bearings, as given in [22]
fi = Aa * muo * cos( theta ) * ( n * i ./ ( 1 ./ mur + 2 .* s0 ) ) .^ 2;

% Force-current constant is the slope (first derivative) at the operating point current

```

% find the index of the operating point current for the top coils

index = find(i >= it0);

cnt = index(1);

% calculate the slope around the operating point for the top coils

kti = (fi(cnt + 1) - fi(cnt - 1)) / (i(cnt + 1) - i(cnt - 1));

% find the index of the operating point current for the bottom coils

index = find(i >= ib0);

cnt = index(1);

% calculate the slope around the operating point for the bottom coils

kbi = (fi(cnt + 1) - fi(cnt - 1)) / (i(cnt + 1) - i(cnt - 1));

%%
 %%% FORCE-POSITION CALCULATIONS %%%
 %%%
 %

% The current value is held constant at the operating point value, and the force of
 % the bearing is calculated for each of the position values in the array "s"

%
 %%%

% Force of magnetic bearings, as given in [22]

% for the top coils

fts = Aa * muo * cos(theta) * (n * it0 ./ (1 ./ mur + 2 .* s)).^ 2;

% for the bottom coils

fbs = Aa * muo * cos(theta) * (n * ib0 ./ (1 ./ mur + 2 .* s)).^ 2;

% Force-position constant is the slope (first derivative) at the operating point

% find index of the operating point position

index = find(s >= s0);

cnt = index(1);

% calculate the slope around the operating point for the top coils

kts = (fts(cnt + 1) - fts(cnt - 1)) / (s(cnt + 1) - s(cnt - 1));

% calculate the slope around the operating point for the bottom coils


```
kbs = ( fbs( cnt + 1 ) - fbs( cnt - 1 ) ) / ( s( cnt + 1 ) - s( cnt - 1 ) );
```

```
%      Display results on the screen
```

```
kti
```

```
kbi
```

```
kts
```

```
kbs
```

APPENDIX B—CONTROLLER DESIGN PROGRAM

```
% ambroots.m
%
% John Wiggins, 09OCT97
%
% an .m file to design a PID controller for the AMBER project
% and calculate step and frequency responses.
%
% This program creates and tests a PID controller for the magnetic bearing system.
% The user chooses the two zeros of the controller—the poles are preset. The
% program then displays the root locus of the system and allows the user to select
% the controller gain by choosing points on the root locus. Once the gain has been
% selected, the program calculates and displays the step response and frequency
% response of the system.

%%%%%%%%%%%%% CREATION OF THE BEARING PLANT MODEL %%%%%%%%%%%%%%

% System parameters:
m_ = 0.907;           % rotor mass (kg)
r_ = 0.6;             % coil resistance (Ohms)
l_ = 0.0015;          % coil inductance (H)
ki_ = 1.3291;         % bottom coil force-current constant (N/A)
kip_ = 6.2026;        % top coil force-current constant (N/A)
ks_ = 758;            % bottom coil force-displacement constant (N/m)
ksp_ = -18953;        % top coil force-displacement constant (N/m)
ko_ = 787;            % sensor gain (V/m)

% Plant transfer function:
bearingN = ( kip_ + ki_ ) / ( m_ * l_ );           % Numerator
bearingD = conv( [1 ( r_ / l_ )], [1 0 ( ( ksp_ + ks_ ) / m_ )] ); % Denominator

%%%%%%%%%%%%% CONTROLLER ZERO SELECTION %%%%%%%%%%%%%%

% Having finished all of the necessary preliminary calculations, the program now asks
% the user to design a PID controller. The user need only select the locations of the two
% controller zeros—the program will then determine the behavior of the closed-loop
% system. Controller design is largely a trial and error process, especially in the final
% stages, so it is extremely useful to have a program that will do the tedious
```

```

% calculations.

% Input the controller zeros:
zero1 = input( 'First controller zero: ' );
zero2 = input( 'Second controller zero: ' );

% Find the controller and system transfer functions
numc = ko_ * conv( [ 1 zero1 ], [ 1 zero2 ] );      % Chosen controller zeros
denc = conv( [ 1 0 ], [ 1 2000 ] );                % The two controller poles are
preset
numt = conv( numc, bearingN );                      % System numerator
dent = conv( denc, bearingD );                      % System denominator

% Plot the root locus
figure;
rlocus( numt, dent );
axis( [ -300 100 -300 300 ] );

% With the shape of the root locus determined by the user's choice of controller zeros,
% only the overall system gain remains to be chosen. The user graphically selects the
% desired location of the poles along the root locus. The program then indicates the
% value of the system gain for the chosen poles, and allows the user to select a new set
% of poles or to run a system simulation with the selected poles.

% Choose desired roots:
continue = 'y';
cnt = 0;
while continue == 'y',
    cnt = cnt + 1;
    snt = num2str( cnt );
    [ k, poles ] = rlocfind( numt, dent );
    gtext( snt );    % Mark each pair of roots
    gtext( snt );
    ks = num2str( k );
    gtext( [ snt, ' K = ', ks ] ); % Indicate k value for selected roots
    continue = input( 'Continue?', 's' );
end

% Title the plot with the chosen zeros
sero1 = num2str( zero1 );
sero2 = num2str( zero2 );

```

```
title( [ 'PID-compensated root locus. Controller zeros: ', sero1, ', ', sero2 ] );
```

```
%%%%%%%%%%%% STEP RESPONSE %%%%%%%%%%
```

```
% Matlab again provides a single function that will determine the step response of a
% sytem. The closed loop transfer function is calculated and the length of the simulation
% is set before running the simulation.
```

```
[ num, den ] = cloop( k * numt, dent );      % Closed loop transfer function
t = [ 0 : 0.001 : 0.5 ];                    % simulation length (seconds)
```

```
% Run and plot the simulation:
```

```
figure;
```

```
step( num, den, t );
```

```
title( [ 'Step response for controller zeros ', sero1, ', ', sero2, 'and k = ', ks ] );
```

```
%%%%%%%%%%%% FREQUENCY RESPONSE %%%%%%%%%%
```

```
% Matlab also provides a function to calculate the frequency response of a system.
```

```
% Set the frequency range:
```

```
w = logspace( 0, 4, 1000 );
```

```
% Find and plot the frequency response for the closed-loop magnetic bearing system,
```

```
% with the controller chosen by the user:
```

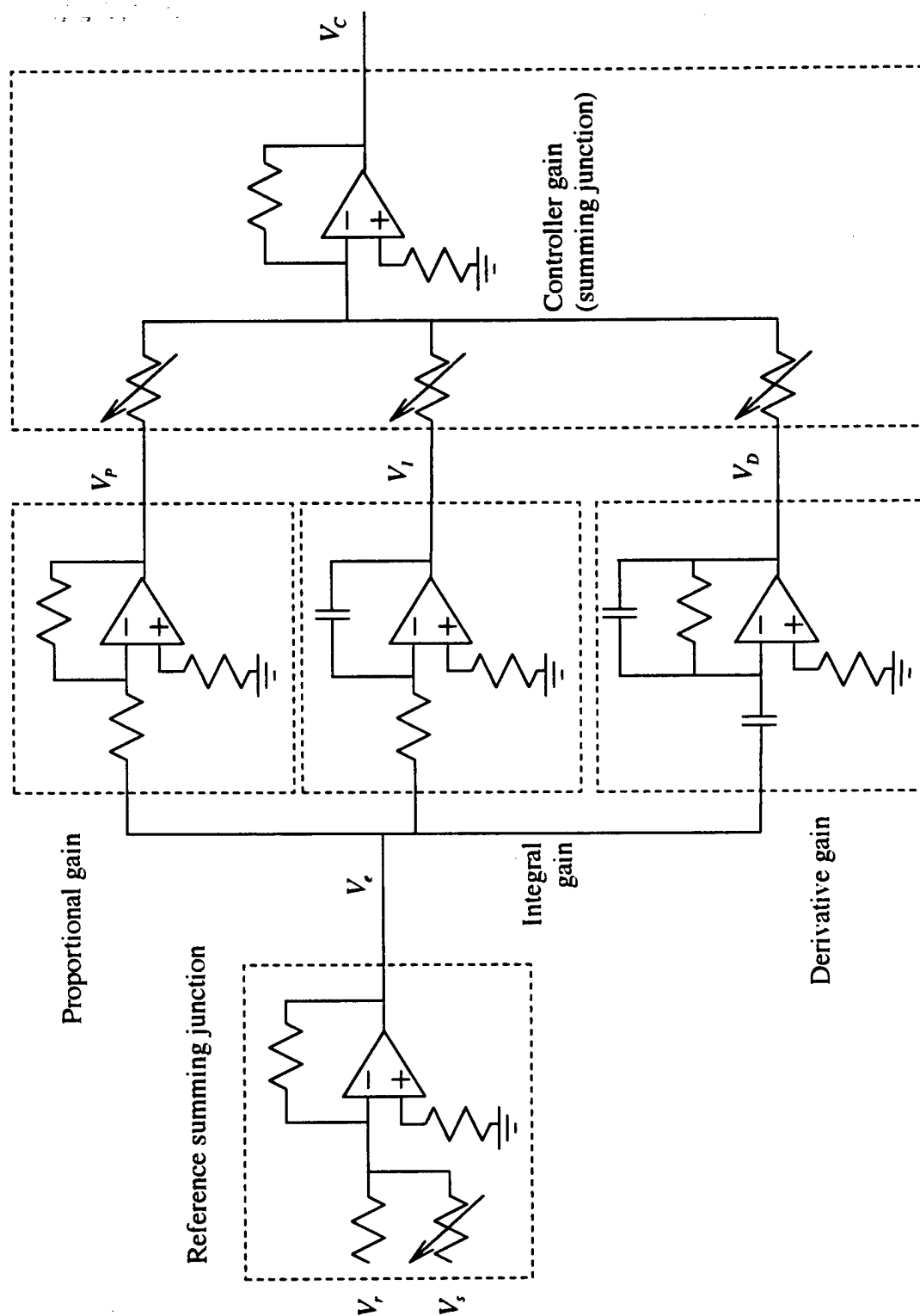
```
figure;
```

```
bode( k * numt, dent, w );
```

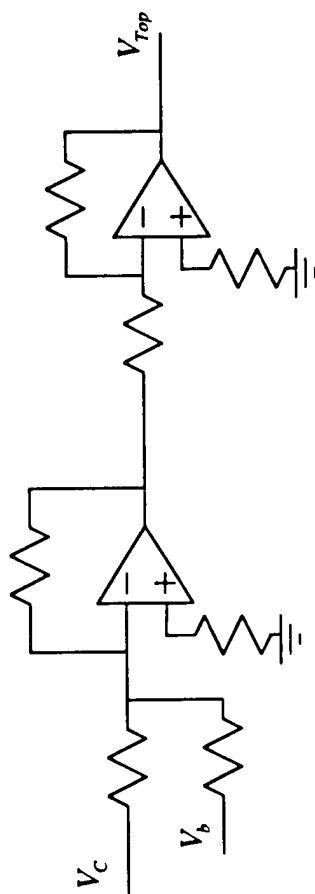
APPENDIX C—MAGNETIC BEARING CONTROLLER CIRCUIT

The circuit schematics for the magnetic bearing PID controller are shown in the following two pages. The controller is divided into four blocks, each centered around an op amp. The reference summing junction provides inputs for a position signal, V_s , which is provided by the position sensors around the shaft, and a reference signal, V_r , which is zero under standard conditions, and is driven by the DSP output for noise control operation. The error between these two voltages, V_e , is fed into three op amp circuits, which provide a proportional gain, and integral gain, and a derivative gain. The output voltages of these three circuits, V_P , V_I , and V_D , are combined and amplified by a common gain in the final summing junction. The output of this summing junction, V_C , is the true controller output, and is then passed to the differential drive circuits.

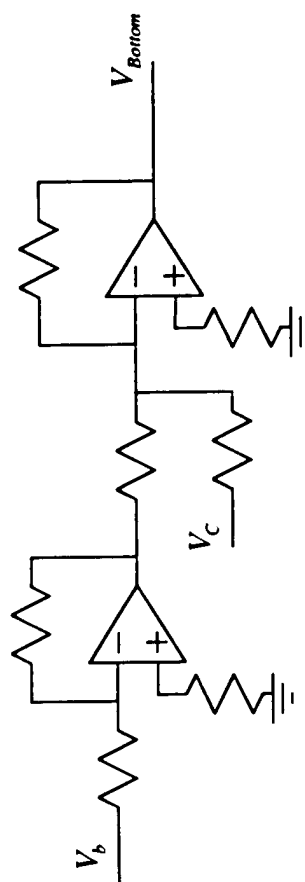
The differential drive circuits modify the control output voltage to produce two different drive voltages: one for the top channel, and one for the bottom channel (or for opposing horizontal channels). An equal bias voltage (V_b) is applied to both channels to increase the strength of the magnetic field of the bearings. The top channel driver adds the control output voltage (V_C) to the bias voltage, yielding a total drive voltage for the top coils. The bottom channel subtracts the control output voltage from the bias voltage to arrive at a drive voltage for the bottom coils. The output of each differential drive channel is fed to a power amplifier to provide the necessary current for the magnetic bearings.



Top channel differential driver



Bottom channel differential driver



APPENDIX D—AMPLIFIER SPECIFICATIONS

SA50 Pulse Width Modulating Power Amplifier—Apex Microtechnology Corporation

ABSOLUTE MAXIMUM RATINGS

Supply Voltage, +V _s	80V
Output Current, peak	7A
Logic Supply Voltage, V _{cc}	16V
Power Dissipation, internal	120W ¹
Temperature, pin solder—10 s	300°C
Temperature, junction ³	150°C
Temperature, storage	-65 to +150°C
Operating Temperature Range, case	-65 to +150°C

SPECIFICATIONS

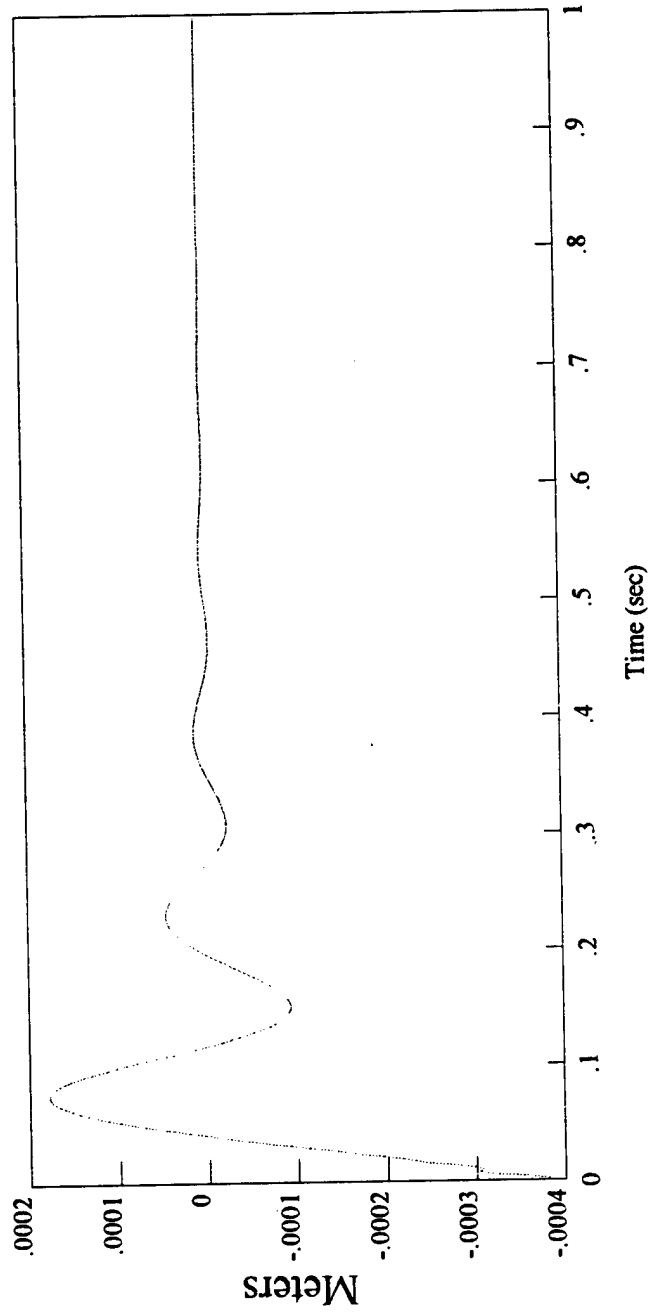
Parameter	Test Conditions ²	MIN	TYP	MAX	UNITS
ANALOG INPUT VOLTAGES					
Motor A, B=50% Duty Cycle	V _{cc} =12V		6		V _{DC}
Motor A = 100% Duty Cycle			8		V _{DC}
Motor B = 100% Duty Cycle			4		V _{DC}
OUTPUT					
V _{ds} voltage, each MOSFET	I _{ds} =5A		1.25	1.8	V _{DC}
Total R _{on} , both MOSFETs			0.5		Ω
Efficiency, 5A output	+V _s =80V		97		%
Switching frequency		40	45	50	Khz
Current, continuous		5			A
Current, peak	t=100 msec	7			A
Switching characteristics⁴					
Rise time	+V _s =28V, V _{cc} =12V I _c =2A		36	54	nS
Fall time			170	250	nS
Dead time			100		nS
POWER SUPPLY					
+V _s voltage	+V _s current=Load			80	V _{DC}
V _{cc} voltage	current	9	12	16	V _{DC}
V _{cc} current	V _{cc} =12V _{DC}		15	20	mA
THERMAL³					
Resistance, junction to case	Full temp range		2.0		°C/W
Resistance, junction to air			3.0		°C/W
Temperature range, case		-25		+85	°C

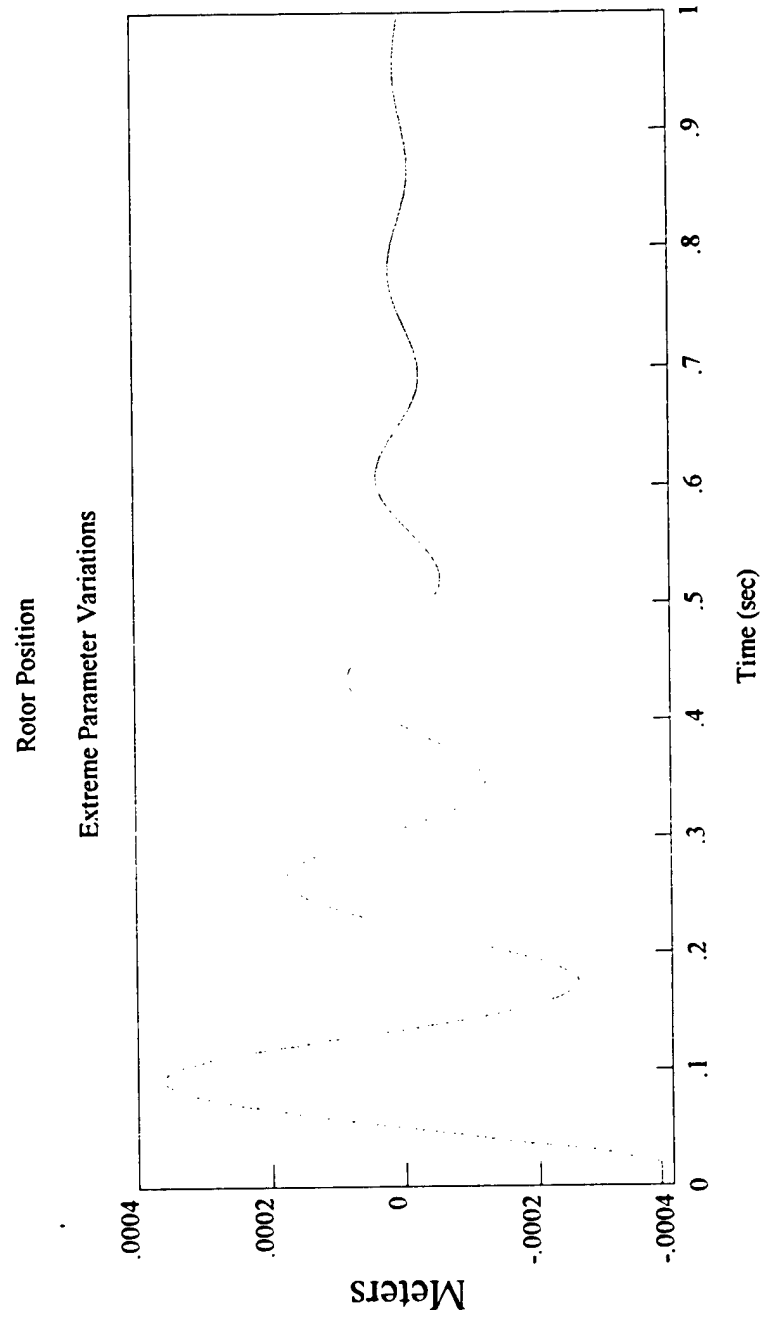
NOTES:

- Each of the two active output transistors can dissipate 60W.
- Unless otherwise noted, T_c=25°C, V_{cc}=12V_{DC}
- Long term operation at the minimum junction temperature will result in reduced product life. Derate internal power dissipation to achieve high MTTF. For guidance, refer to the heatsink data sheet
- Guaranteed but not tested.

APPENDIX E—MAGNETIC BEARING SIMULATION

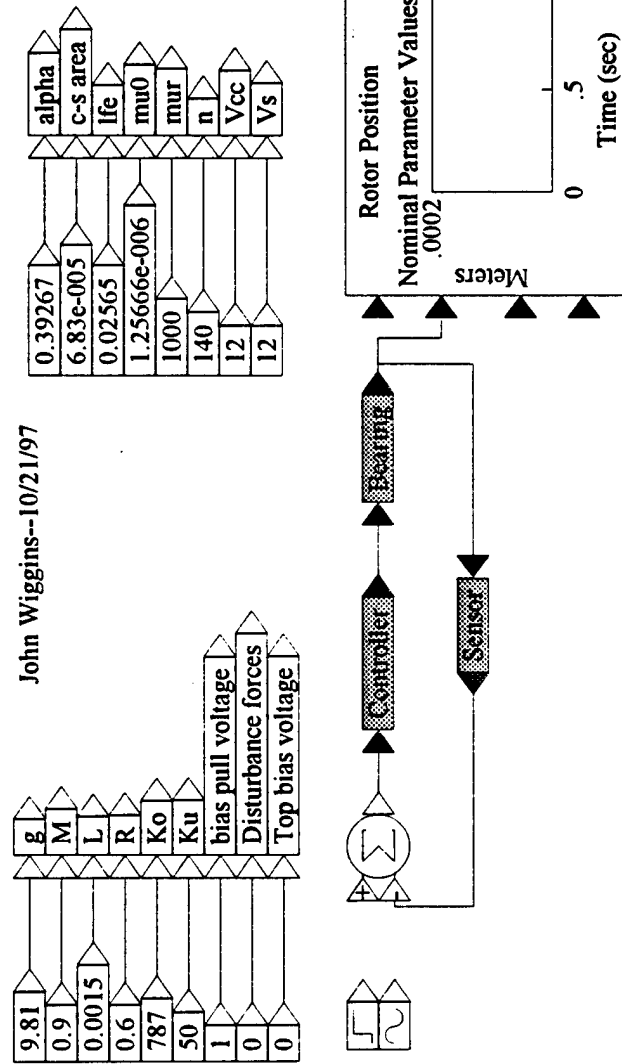
Rotor Position
Nominal Parameter Values





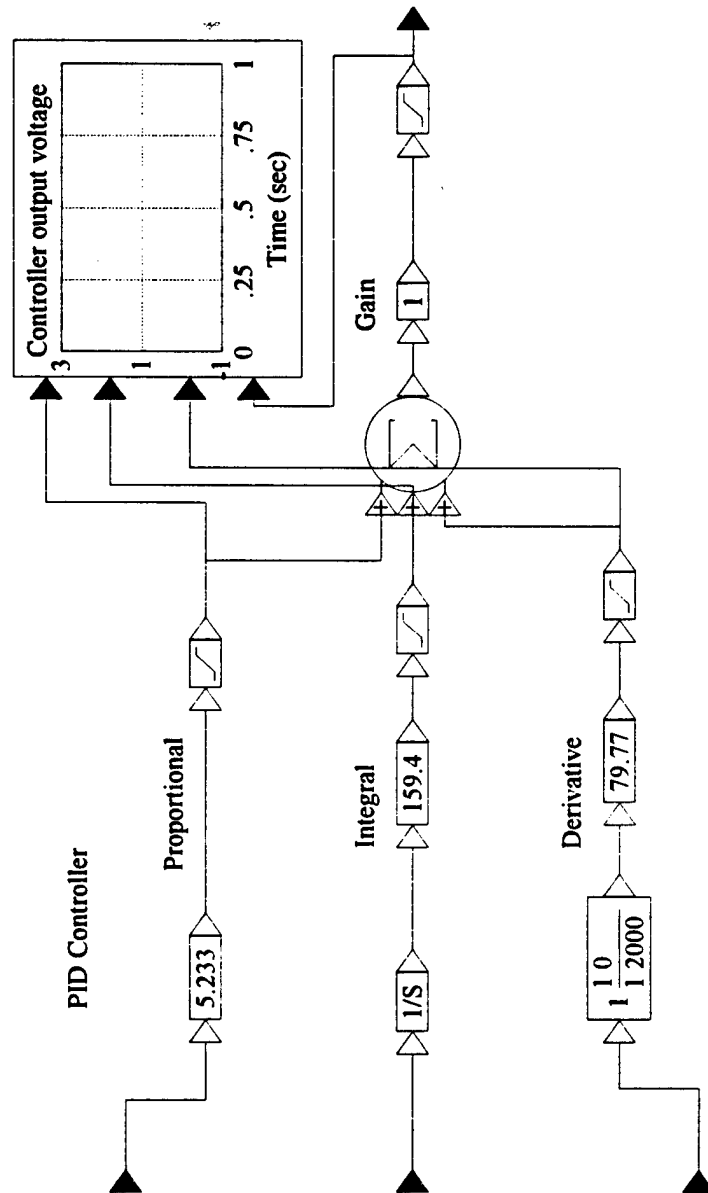
Active Magnetic Bearing System

John Wiggins--10/21/97

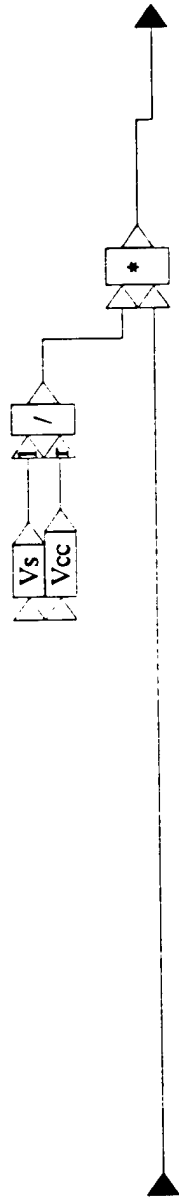


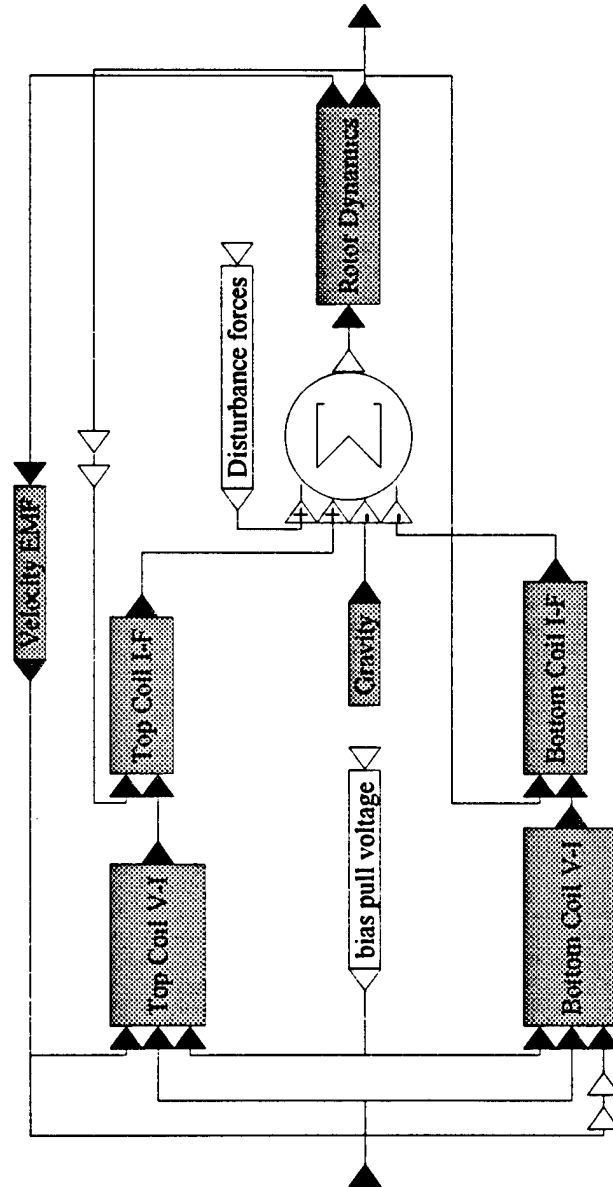
VisSim/32 -AMBYRFV.VSM::Controller

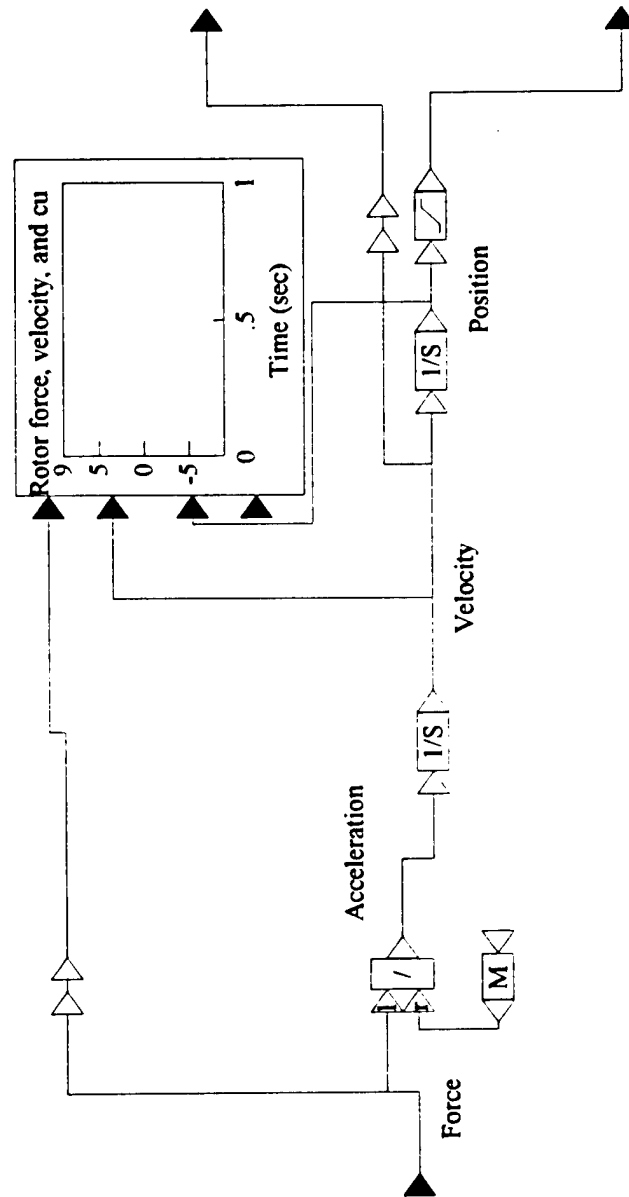




VisSim/32 - AMBYRFV.VSM::Controller.Amplifier



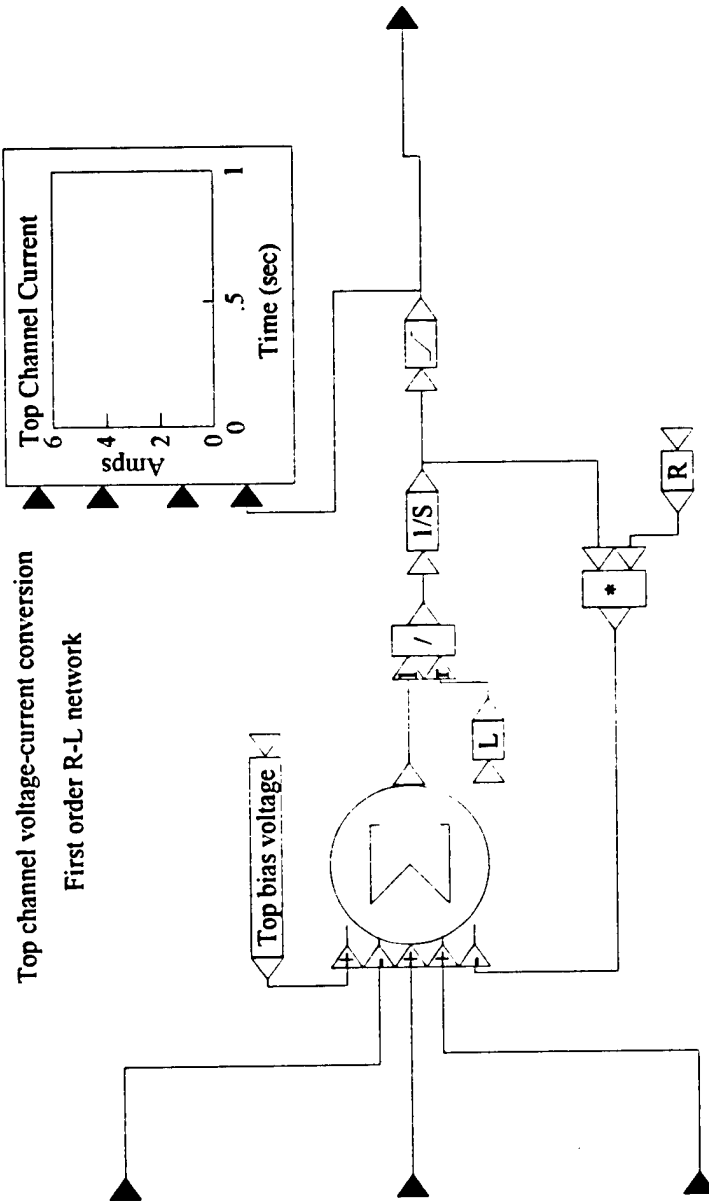




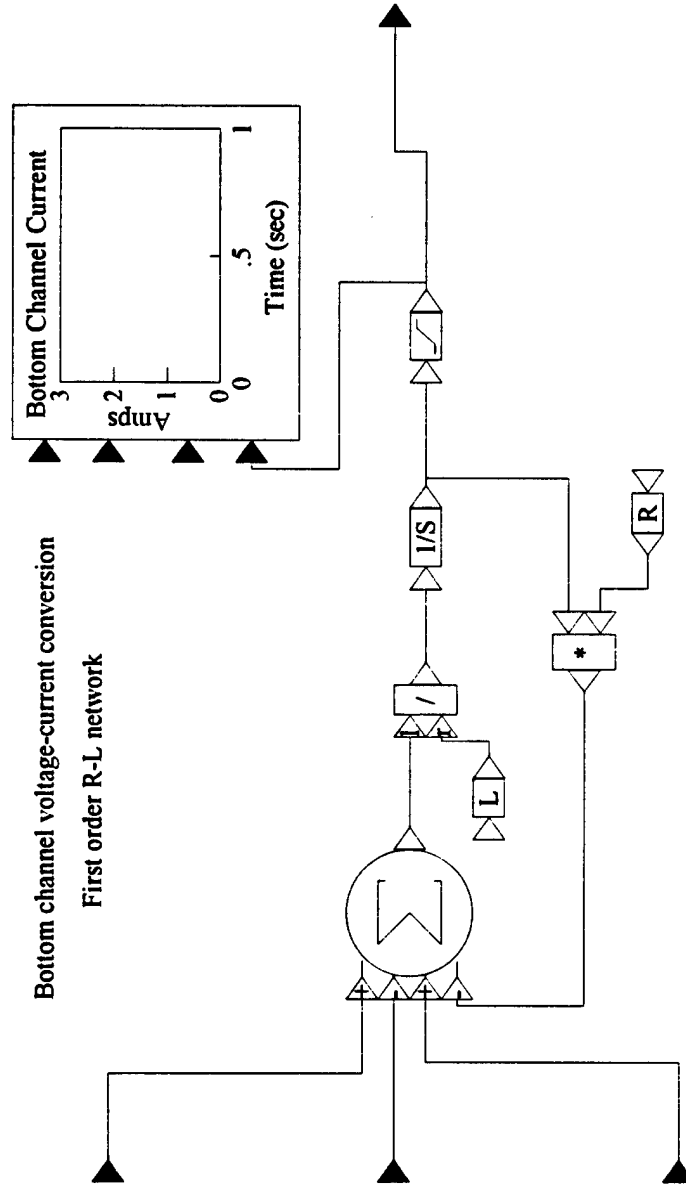
Rotor mechanics

VisSim/32 -AMBYRFV.VSM::Bearing.Gravity

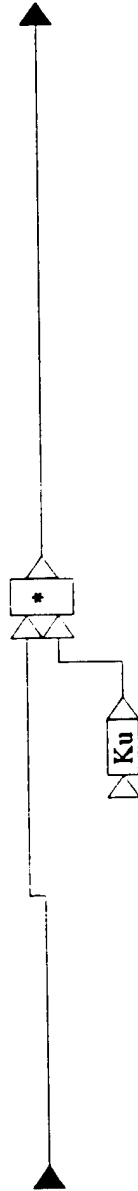


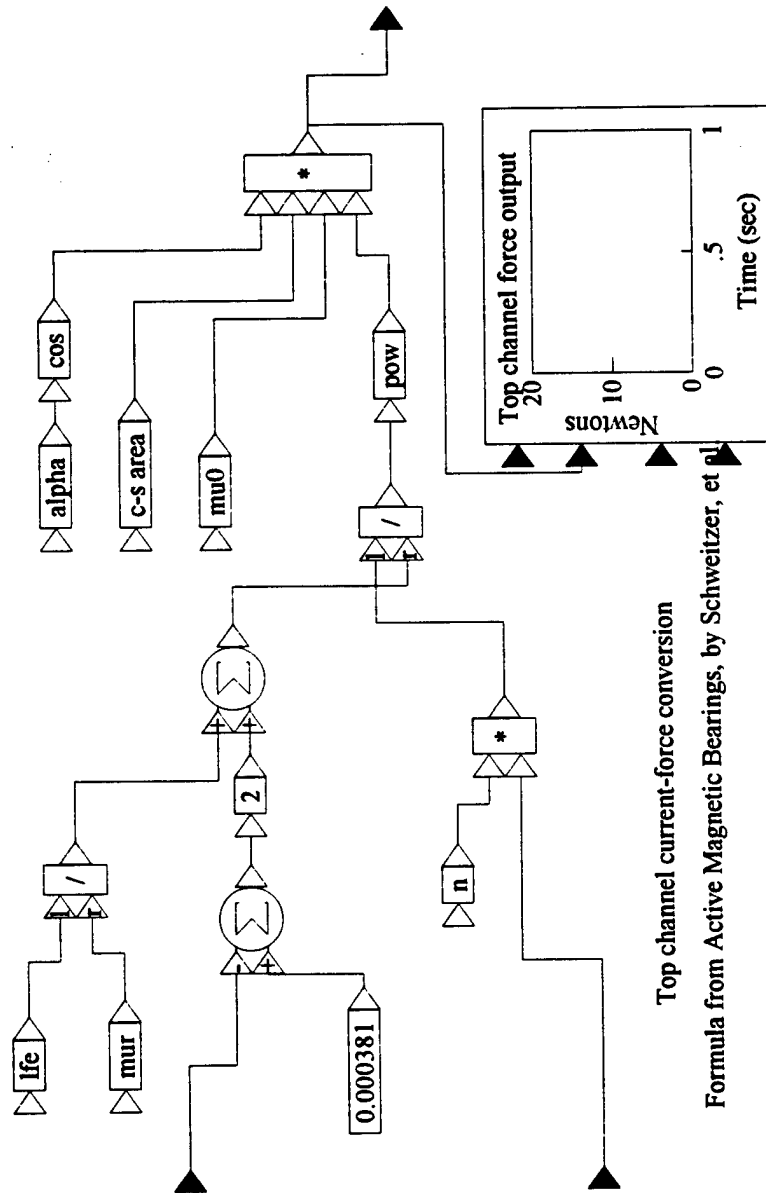


label



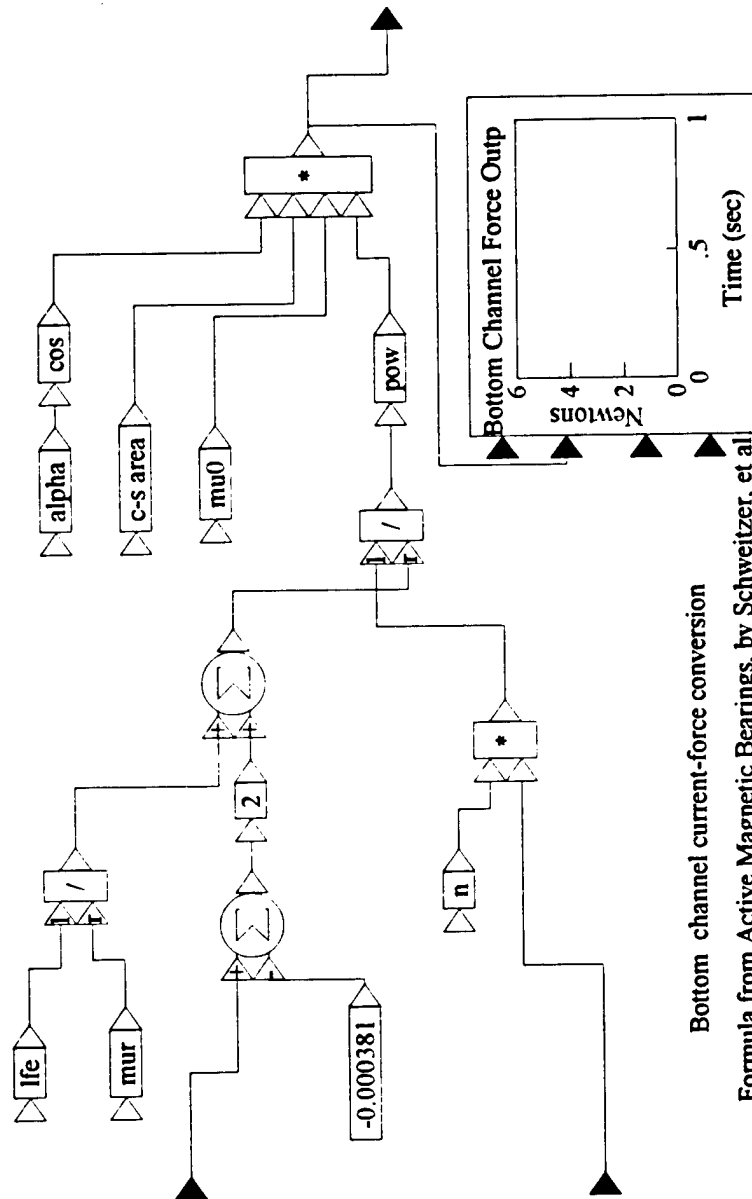
VisSim/32 -AMBYRFV.VSM::Bearing_Velocity EMF





Top channel current-force conversion

Formula from Active Magnetic Bearings, by Schweitzer, et al



Bottom channel current-force conversion

Formula from Active Magnetic Bearings, by Schweitzer, et al

VisSim/32 -AMBYRFV.VSM::Sensor



APPENDIX F—NOISE CONTROL SIMULATION

```
% ANR.m
%
% An active sound control simulation for
% the AMBER apparatus
%
% by John Wiggins
% 10/30/97
%
% This file provides an interface for the .mex file filteredx.c, and defines certain
% constants for the simulated system:
%
% t          determines the length of the simulation
%
% numTaps    sets the number of taps in the digital filter
%
% mu         is the step size for the adaptive filter algorithm
%
% refFreq    is a vector containing the reference (feed-forward) frequencies
%            for the system
%
% distFreq   is a vector containing the disturbance frequencies for the system
%
% priA       These are the numerator and denominator of the path transfer
%            function
%
% priB       from the noise sources (primary and secondary) to the sensor
%            microphone

clear; % Free up all of Matlab's memory for this program

% amberh.m defines the discrete-time AMBER transfer function rvsN / rvsD
% from the reference-voltage input (connected to the DSP output) to
% the sound output, as well as the sampling time for the system (Ts). The text
% of "amberh.m" is located at the end of this appendix.
amberh;

% Set simulation parameters
t = 0 : Ts : 20;          % Simulation length
maxI = length( t );
refFreq = [100];         % Reference frequency
```

```

distFreq = [95];           % Disturbance frequencies
pathB = [0 0 0 0 0 0 1];   % Primary and secondary path transfer function
pathA = [1 0 0 0 0 0 0];
plantB = rvsNd;             % Magnetic bearing actuator transfer function
plantA = rvsDd;

% Filter and LMS parameters
numTaps = 100;              % Length of adaptive filter
mu = 1e-7;                  % Convergence step size
filtB = rvsNd;              % Secondary path model
filtA = rvsDd;

% Create reference signal
numRefs = length( refFreq );
refAmp = 1 / numRefs;
refSound = zeros( 1, maxI );
for n = 1 : numRefs,
    refSound = refSound + refAmp * sin( refFreq(n) * 2 * pi * t );
end

priSound = refSound;

% Create disturbance sound
numDists = length( distFreq );
distAmp = 1 / numDists;
distSound = zeros( 1, maxI );
for n = 1 : numDists,
    distSound = distSound + distAmp * sin( distFreq(n) * 2 * pi * t );
end

% Run the heavy part of the simulation in a mex-file, so that it runs more quickly.
% This is a call to filteredx.c. Program flow enters the function "mexFunction" in
% filteredx.c from this point.
[errorSig, priSound, secSound, secControl, secActuating, filtSig, tap] = ...
filteredx( refSound, pathB, pathA, distSound, plantB, plantA, filtB, filtA, numTaps, mu);

% When filteredx.c is done running, it will return all of the simulation results in the
% variables "errorSig," "priSound," "secSound," etc. Standard Matlab functions are then
% used to display the results of the simulation.

% First, the time-domain results are plotted. For systems with pure tone signals, these

```

```
% results are most meaningful.
```

```
result = errorSig;
```

```
figure;
```

```
plot( t, result );
```

```
title( ['Figure 12-c Output of System With ', num2str( refFreq ), ...  
' Hz Reference Signal'] );
```

```
xlabel( 'time' );
```

```
% Next, the frequency-domain results are plotted. When broadband noise is used for  
% the disturbance sound, the time-domain results become impossible to interpret.  
% Frequency-domain results, however, will show the changes in the component signal  
% amplitudes clearly.
```

```
% Plot steady-state frequency-domain results:
```

```
% The frequency-domain results are calculated by taking the Fourier transform of the  
% time-domain results.
```

```
index = find( t == 5 );
```

```
x = result( 1, index : length( t ) - 1 );
```

```
y = fft( x, 1024 );
```

```
m = abs( y ) * 2 / length( y );
```

```
p = unwrap( angle( y ) );
```

```
f = ( 0 : 511 ) * 2;
```

```
figure;
```

```
plot( f, m( 1 : 512 ) );
```

```
title( ['DFT of Steady-State System Output'] );
```

```
xlabel( 'freq' );
```

```
ylabel( 'magnitude' );
```

/******

filteredx.c by John Wiggins Dec 10, 1997

A .mex routine called by ANR.m as part of a simulation of an active noise control scheme

A call to this function must have the following format:

filteredx(refSound, pathB, pathA, distSound, plantB, plantA, filtB, filtA, numTaps, mu)

refSound: array (row vector) containing the precalculated values of the reference signal over the full period of time to be simulated

pathB: numerator of the difference equation for the sound path

pathA: denominator of the difference equation for the sound path

pathB and pathA must be arrays (row vectors) of the same length

distSound: array (row vector) containing the precalculated values of the disturbance sound present at the error mic. Must be the same length as refSound

plantB: numerator of the difference equation for the secondary sound actuator

plantA: denominator of the difference equation for the secondary sound actuator
actB and actA must be arrays (row vectors) of the same length

filtB: numerator of the difference equation for the secondary path model

filtA: denominator of the difference equation for the secondary path model

numTaps: number of taps. Length of the weight vector of the digital filter

mu: step size for LMS algorithm

The optional return vector is of the form:

[errorSig, priSound, secSound, secControl, secActuating, filtSig, tap]

If there are fewer arguments present, those present will be assigned values in the order of precedence above--eg, if there are two left hand arguments, the leftmost one will be assigned errorSig and the next one will be assigned secSound. All return arguments are row vectors.

errorSig: sum of the primary, secondary, and disturbance sounds at the error mic.

priSound: the primary sound at the error microphone. Same length as refSound

secSound: the secondary sound at the error mic. Same length as refSound

secControl: output of the feedforward filter. Same length as refSound

secActuating: output of the actuator. Same length as refSound
 tap: the final value of the weight vector for the adaptive filter

```

*****/

// Header files:
#include      "mex.h"
#include      "filteredX.h"
// The text of the file filteredX.h, which was written as a portion of filteredX.c, has been
// included here:

/*****

filteredX.h

Header file for filteredX.c--part of an active noise control simulation.

*****/

// Definition of constants involving input variables:
enum
{
    RHA_refSound      = 0,    // reference sound
    RHA_pathB,          // z-domain primary path numerator
    RHA_pathA,          // z-domain primary path denominator
    RHA_distSound,      // disturbance sound
    RHA_dAmberN,         // z-domain AMBER numerator
    RHA_dAmberD,         // z-domain AMBER denominator
    RHA_filtB,          // z-domain x filter numerator
    RHA_filtA,          // z-domain x filter denominator
    RHA_numTaps,        // number of taps
    RHA_mu,             // convergence step size
    E_nrhs              // Total expected number of right-hand side
arguments
};

// Definition of constants involving output variables
enum
{
    LHA_errorSig      = 0,    // error signal
    LHA_priSound,      // primary sound

```

```

    LHA_secSound,      // secondary sound
    LHA_secControl,    // secondary control signal (filter output)
    LHA_secActuating,  // secondary actuating signal (AMBER sound output)
    LHA_filtSig,       // filtered x signal
    LHA_tap,           // final weight vector values
    E_nlhs             // Total expected number of left-hand side arguments
};

```

```
// Function prototypes
```

```
void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[] );
```

```
void MyANR( double *refSound, long pathNB, double *pathB, long pathNA, double
    *pathA, double *priSound, double *secControl, long amberNB, double
    *dAmberN, long amberNA, double *dAmberD, double *secActuating, double
    *secSound, double *distSound, double *errorSig, long filtNB, double *filtB, long
    filtNA, double *filtA, double *filt, double *tap, long numTaps, double mu, long
    maxI );
```

```
//      END OF FILTEREDX.H
//      RESUME FILTEREDX.C
```

```

/*****
*****      mexFunction      *****/
*****
*
*      The gateway routine for the .mex file. This handles the call
*      from and return to the .m file. Retrieves pointers to and/or
*      values of workspace variables, and assigns return value pointers
*      to return workspace variables. While rather long, this function is
*      only an interface with Matlab. The simulation is run entirely in the
*      function "MyANR," which follows this function
*****/

```

```

void mexFunction(  int      nlhs,
                   mxArray  *plhs[],
                   int      nrhs,
                   const mxArray *prhs[] )
{
    long  amberNB,    // order of the AMBER numerator

```

```

    amberNA,    // order of the AMBER denominator
    pathNB,     // order of the primary and secondary path numerator
    pathNA,     // order of the primary and secondary path denominator
    filtNB,     // order of the x filter numerator
    filtNA,     // order of the x filter denominator
    maxI,       // number of points in and all signal vectors
    priI,       // number of points in the primary sound vector
    distI,      // number of points in the disturbance sound vector
    numTaps,    // length of the filter weight vector
    cnt;

double *dAmberN, // numerator of the z-domain AMBER transfer function
       *dAmberD, // denominator of the z-domain AMBER transfer function
       *pathB,   // numerator of the path transfer function
       *pathA,   // denominator of the path transfer function
       *filtB,   // numerator of the x filter
       *filtA,   // denominator of the x filter
       *refSound, // pointer to the reference signal vector
       *priSound, // pointer to the primary sound vector
       *secControl, // pointer to the secondary control signal vector
       *secActuating, // pointer to the secondary actuating signal vector
       *secSound, // pointer to the secondary sound vector
       *distSound, // pointer to the disturbance sound vector
       *errorSig, // pointer to the error signal vector
       *tap,      // pointer to the weight vector
       *filt,     // pointer to the filtered reference signal vector
       mu;        // step size for the LMS algorithm

mxArray *errorSigArray, // output variables
        *priSoundArray,
        *secSoundArray,
        *secControlArray,
        *secActuatingArray,
        *tapArray,
        *filtArray;

// A little problem catching:
// Check for expected number of input arguments
if ( nrhs != E_nrhs || nlhs > E_nlhs )
{
    mexErrMsgTxt( "\nImproper number of arguments" );
}

```

```

        return;
    }

    // Ensure all vectors are row vectors
    for ( cnt = 0; cnt < nrhs; cnt++ )
    {
        if ( mxGetM( prhs[cnt] ) != 1 || mxGetN( prhs[cnt] ) < 1 )
        {
            mexErrMsgTxt( "\nAll inputs must be row vectors" );
            return;
        }
    }

    // Check length of input vectors for compatibility
    maxI = mxGetN( prhs[RHA_refSound] );
    distI = mxGetN( prhs[RHA_distSound] );
    if ( distI != maxI )
    {
        mexErrMsgTxt( "\nSignal vectors must all have the same length" );
        return;
    }

    // Get transfer function orders:
    amberNB = mxGetN( prhs[RHA_dAmberN] );
    amberNA = mxGetN( prhs[RHA_dAmberD] );
    pathNB = mxGetN( prhs[RHA_pathB] );
    pathNA = mxGetN( prhs[RHA_pathA] );
    filtNB = mxGetN( prhs[RHA_filtB] );
    filtNA = mxGetN( prhs[RHA_filtA] );

    // Get pointers to all of the arguments
    refSound = mxGetPr( prhs[RHA_refSound] );
    pathB = mxGetPr( prhs[RHA_pathB] );
    pathA = mxGetPr( prhs[RHA_pathA] );
    distSound = mxGetPr( prhs[RHA_distSound] );
    dAmberN = mxGetPr( prhs[RHA_dAmberN] );
    dAmberD = mxGetPr( prhs[RHA_dAmberD] );
    filtB = mxGetPr( prhs[RHA_filtB] );
    filtA = mxGetPr( prhs[RHA_filtA] );
    mu = *mxGetPr( prhs[RHA_mu] );
    numTaps = *mxGetPr( prhs[RHA_numTaps] );

```



```

// Allocate memory for output variables:
errorSigArray = mxCreateDoubleMatrix( 1, maxI, mxREAL );
if ( !errorSigArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
errorSig = mxGetPr( errorSigArray );

priSoundArray = mxCreateDoubleMatrix( 1, maxI, mxREAL );
if ( !priSoundArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
priSound = mxGetPr( priSoundArray );

secSoundArray = mxCreateDoubleMatrix( 1, maxI, mxREAL );
if ( !secSoundArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
secSound = mxGetPr( secSoundArray );

secControlArray = mxCreateDoubleMatrix( 1, maxI, mxREAL );
if ( !secControlArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
secControl = mxGetPr( secControlArray );

secActuatingArray = mxCreateDoubleMatrix( 1, maxI, mxREAL );
if ( !secActuatingArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
secActuating = mxGetPr( secActuatingArray );

```

```

filtArray = mxCreateDoubleMatrix( 1, maxI, mxREAL );
if ( !filtArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
filt = mxGetPr( filtArray );

tapArray = mxCreateDoubleMatrix( 1, numTaps, mxREAL );
if ( !tapArray )
{
    mexErrMsgTxt( "\nOut of memory" );
    return;
}
tap = mxGetPr( tapArray );

// Call the computation routine
MyANR( refSound, pathNB, pathB, pathNA, pathA, priSound, secControl,
        amberNB, dAmberN, amberNA, dAmberD, secActuating, secSound,
        distSound, errorSig, filtNB, filtB, filtNA, filtA, filt, tap, numTaps, mu,
        maxI );

// Assign output variables as necessary
if ( nlhs >= 1 )
    plhs[LHA_errorSig] = errorSigArray;

if ( nlhs >= 2 )
    plhs[LHA_priSound] = priSoundArray;

if ( nlhs >= 3 )
    plhs[LHA_secSound] = secSoundArray;

if ( nlhs >= 4 )
    plhs[LHA_secControl] = secControlArray;

if ( nlhs >= 5 )
    plhs[LHA_secActuating] = secActuatingArray;

if ( nlhs >= 6 )
    plhs[LHA_filtSig] = filtArray;

```

```

        if ( nlhs >= 7 )
            plhs[LHA_tap] = tapArray;
    }

```

```

/*****
***** MyANR *****/
*****
*
*   Simulates an active noise control scheme using the adaptive least-mean-squares
*   algorithm. A reference signal is given, along with the order and step size of the
*   filter and the z-domain transfer function of the actuator (AMBER). The reference
*   signal is filtered through a FIR filter, whose weight vector is modified at each
*   iteration with the LMS algorithm. The output of the filter is the secondary control
*   signal (secControl), which is input to the AMBER system, which is the actuator.
*   secSound, priSound, and distSound (the disturbance sound at the microphone) are
*   added to produce the error signal seen by the LMS algorithm. Recalculation of
*   the weight vector is the final step of each iteration.
*****/

```

```

void MyANR( double    *refSound,
              long     pathNB,
              double   *pathB,
              long     pathNA,
              double   *pathA,
              double   *priSound,
              double   *secControl,
              long     amberNB,
              double   *dAmberN,
              long     amberNA,
              double   *dAmberD,
              double   *secActuating,
              double   *secSound,
              double   *distSound,
              double   *errorSig,
              long     filtNB,
              double   *filtB,
              long     filtNA,
              double   *filtA,
              double   *filt,

```

```

double      *tap,
long        numTaps,
double      mu,
long        maxI )
{

    long      i,
             n;

    // Loop through, computing for each element (discrete time period):
    for ( i = 0; i < maxI; i++ )
    {

        /*****
        *      Calculation of the primary sound                      *
        *      This is the evaluation of the discrete time transfer function      *
        *      pathB / pathA                                          *
        *****/
        for ( n = 0; n < pathNB; n++ )
        {
            // avoid using negative array indices
            if ( n > i )
                break;

            priSound[i] += pathB[n] * refSound[i - n];
        }
        for ( n = 1; n < pathNA; n++ )
        {
            // avoid using negative array indices
            if ( n > i )
                break;

            priSound[i] -= pathA[n] * priSound[i - n];
        }
        priSound[i] /= pathA[0];

        /*****/
    }
}

```

```

/*****
*      Calculation of the DSP output (ANR filter)
*      The weighted average of the past 100 (or numTaps) inputs is
*      evaluated. The weights are stored in the array "tap"
*****/
    for ( n = 0; n < numTaps; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        // a FIR filter of order numTaps
        secControl[i] += tap[n] * refSound[i - n];
    }

/*****/

/*****
*      Calculation of the sound output of the magnetic bearings
*      This is the evaluation of the discrete time transfer function
*      amberB / amberA
*****/
    for ( n = 0; n < amberNB; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        secActuating[i] += dAmberN[n] * secControl[i - n];
    }
    for ( n = 1; n < amberNA; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        secActuating[i] -= dAmberD[n] * secActuating[i - n];
    }
    secActuating[i] /= dAmberD[0];

```

```

/*****/

/*****
*      Calculation of the secondary sound                      *
*      This is the evaluation of the discrete time transfer function *
*      pathNB / pathNA                                          *
*****/

    for ( n = 0; n < pathNB; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        secSound[i] += pathB[n] * secActuating[i - n];
    }
    for ( n = 1; n < pathNA; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        secSound[i] -= pathA[n] * secSound[i - n];
    }
    secSound[i] /= pathA[0];

/*****/

/*****
*      Calculation of the error signal                          *
*      The primary, secondary, and disturbance sounds are added *
*****/

    errorSig[i] = priSound[i] + secSound[i] + distSound[i];

/*****/

```

```

/*****
*      The filtered-x portion of the adaptive algorithm      *
*      The input signal is filtered by a copy of the secondary path      *
*      transfer functions as stored in filtB / filtA      *
*****/
    for ( n = 0; n < filtNB; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        filt[i] += filtB[n] * refSound[i - n];
    }
    for ( n = 1; n < filtNA; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        filt[i] -= filtA[n] * filt[i - n];
    }
    filt[i] /= filtA[0];
/*****/

/*****/
*      The adaptive algorithm (LMS)      *
*      each of the weights in the weight vector is altered according      *
*      to the formula  $w(n+1) = w(n) + \mu * x * e$       *
*****/
    for ( n = 0; n < numTaps; n++ )
    {
        // avoid using negative array indices
        if ( n > i )
            break;

        // the LMS algorithm:
        tap[n] = tap[n] + mu * filt[i - n] * errorSig[i];
    }
/*****/
}
}

```

```

%   amberh.m
%   by John Wiggins

%   This file calculates three transfer functions for the closed-loop,
%   controlled magnetic bearings:
%   1) reference input to position output—amberSys
%   2) disturbance input to velocity (sound) output—dvsSys
%   3) reference input to velocity (sound) output—rvsSys

% System parameters:
m_ = 0.907;           % rotor mass (kg)
r_ = 0.6;            % coil resistance (Ohms)
l_ = 0.0015;         % coil inductance (H)
ki_ = 1.3291;        % bottom coil force-current constant (N/A)
kip_ = 6.2026;       % top coil force-current constant (N/A)
ks_ = 758;           % bottom coil force-displacement constant (N/m)
ksp_ = -18953;       % top coil force-displacement constant (N/m)
ko_ = 787;           % sensor gain (V/m)
kcp_ = 10625;        % controller proportional gain
kci_ = 318750;       % controller integral gain
kcd_ = 85;           % controller derivative gain
tauinv_ = 2000;      % controller LPF pole

% Open-loop transfer function
%bearingN = 3582;
%bearingD = [1 400 -21732 -8692800];
bearingN = ( kip_ + ki_ ) / ( m_ * l_ );
bearingD = conv( [1 ( r_ / l_ )], [1 0 ( ( ksp_ + ks_ ) / m_ )] );

controllerN = [kcd_ kcp_ kci_];
controllerD = [1 tauinv_ 0];

% input to position transfer function:
amberOpenN = conv( controllerN, bearingN );
amberOpenD = conv( controllerD, bearingD );
amberOpenSys = tf( amberOpenN, amberOpenD );
amberSys = feedback( amberOpenSys, ko_ );

% Closed-loop disturbance-voltage to sound (velocity) transfer function:
dvsForwardN = [( kip_ + ki_ ) 0];
dvsForwardD = conv( [1_ r_], [m_ 0 ( ksp_ - ks_ )] );

```



```

dvsForwardSys = tf( dvsForwardN, dvsForwardD );

dvsFeedbackN = ko_ * controllerN;
dvsFeedbackD = conv( [1 0], controllerD );
dvsFeedbackSys = tf( dvsFeedbackN,dvsFeedbackD );

dvsSys = feedback( dvsForwardSys, dvsFeedbackSys );

% Closed-loop reference-input to sound (velocity) transfer function:
rvsForwardN = conv( controllerN, [( kip_ + ki_ ) 0] );
rvsForwardD = conv( conv( controllerD, [l_ r_ ] ), [m_ 0 -( ksp_ - ks_ )] );
rvsForwardSys = tf( rvsForwardN, rvsForwardD );

rvsFeedbackN = ko_;
rvsFeedbackD = [1 0];
rvsFeedbackSys = tf( rvsFeedbackN, rvsFeedbackD );

rvsSys = feedback( rvsForwardSys, rvsFeedbackSys );

% In addition to the theoretical transfer functions calculated above, this file will
% also create a transfer function from experimental data obtained from
% measurements of the magnetic bearings' frequency response.

% Experimental transfer functions from a curve fit of the frequency response
% zeros:
mz1 = 75.2948;
mz2 = 487.187 + 1222.52i;
mz3 = 487.187 - 1222.52i;
mz4 = 204.037 + 3672.67i;
mz5 = 204.037 - 3672.67i;
mz6 = -395.196 + 6568.8i;
mz7 = -395.196 - 6568.8i;
mz8 = 450.53 + 6598.15i;
mz9 = 450.53 - 6598.15i;
mz10 = -274.493 + 8145.79i;
mz11 = -274.493 - 8145.79i;
mz12 = 7715.47 + 8555.05i;
mz13 = 7715.47 - 8555.05i;

```

```

% poles:
mp1 = -3.00797;
mp2 = -271.886 + 236.997i;
mp3 = -271.886 - 236.997i;
mp4 = -260.762 + 2086.48i;
mp5 = -260.762 - 2086.48i;
mp6 = -10.406 + 3717.81i;
mp7 = -10.406 - 3717.81i;
mp8 = -51.8782 + 6827.74i;
mp9 = -51.8782 - 6827.74i;
mp10 = -56.1041 + 8033.2i;
mp11 = -56.1041 - 8033.2i;
mp12 = -63.017 + 10035.9i;
mp13 = -63.017 - 10035.9i;

% Experimental transfer function:
measN = poly( [mz1 mz2 mz3 mz4 mz5 mz6 mz7 mz8 mz9 mz10 mz11 mz12 mz13] );
measD = poly( [mp1 mp2 mp3 mp4 mp5 mp6 mp7 mp8 mp9 mp10 mp11 mp12 mp13] );
measSys = tf( 2.5e-5 * measN, measD );

% This file also converts the above continuous-time transfer functions to discrete-
% time transfer functions for use in computer simulations.

% Discrete-time converted functions:
Ts = 2.5e-4;

dvsSysd = c2d( dvsSys, Ts, 'zoh' );
[dvsNd, dvsDd] = tfdata( dvsSysd, 'v' );
dvsOrder = length( dvsDd );

rvsSysd = c2d( rvsSys, Ts, 'zoh' );
[rvsNd, rvsDd] = tfdata( rvsSysd, 'v' );
rvsOrder = length( rvsDd );

measSysd = c2d( measSys, Ts, 'zoh' );
[measNd, measDd] = tfdata( measSysd, 'v' );
measOrder = length( measDd );

```

APPENDIX G—SHARC PROCESSOR SPECIFICATIONS

ADSP-21061 SHARC

Analog Devices, October, 1996

Preliminary Data

SUMMARY

- High-performance signal computer for speech, sound, graphics, and imaging
- Super Harvard Architecture Computer (SHARC)—four independent buses for dual data, instructions, and I/O
- 32-bit IEEE floating point computation units multiplier, ALU, and shifter
- 1 Megabit on-chip SRAM memory and integrated I/O peripherals
- Integrated multiprocessing features

KEY FEATURES

- 40 MIPS, 25 ns instruction rate, single cycle instruction execution
- 120 MFLOPS peak, 80 MFLOPS sustained performance
- Dual data address generators with modulo and bit-reverse addressing
- Efficient program sequencing with zero-overhead looping: single cycle loop setup
- IEEE JTAG Standard 1149.1 test access port and on-chip emulation
- 240-lead PQFP package
- Pin-compatible with ADSP-21060 (4 MBit) & ADSP-21062 (2 MBit)
- 5.0 Volt operation

Flexible Data Formats & 40-Bit Extended Precision:

- 32-Bit single-precision & 40-Bit extended precision IEEE floating-point data formats
- 32-Bit fixed-point data format, integer and fractional, with 80-Bit accumulators

Parallel Computations:

- Single-cycle multiply & ALU operations in parallel with dual memory read/writes & instruction fetch
- Multiply with add & subtract for accelerated FFT butterfly computation
- 1024-point complex FFT benchmark: 0.46 msec (18,221 cycles)

1 Megabit Configurable On-Chip SRAM:

- Dual-ported for independent access by core processor and DMA
- Configurable as 32K words data memory (32-Bit), 16K words program memory (48-Bit), or combinations of both up to 1 MBit

Off-Chip Memory Interfacing:

- 4 gigawords addressable (32-bit address)
- Programmable wait state generation, page-mode DRAM support

DMA Controller:

- 6 DMA Channels
- Background DMA Transfers at 40 MHz, in parallel with full-speed processor execution
- Performs transfers between ADSP-21061 internal memory and external memory, external peripherals, host processor, or serial ports

Host Processor Interface:

- Efficient interface to 16- & 32-Bit microprocessors
- Host can directly read/write ADSP-21061 internal memory

Multiprocessing:

- Glueless connection for scalable DSP multiprocessing architecture
- Distributed on-chip bus arbitration for parallel bus connect of up to 6 ADSP-21061s plus host
- 240 MBytes/s transfer rate over parallel bus

Serial Ports:

- Two 40 MBit/s synchronous serial ports
- Independent transmit and receive functions
- 3- to 32-Bit data word width
- μ -law/A-law hardware companding
- TDM multichannel mode
- Multichannel signaling protocol

APPENDIX H—REFERENCE SIGNAL GENERATION PROGRAM

/* **** */

TTRefSig.c

by John Wiggins 11 February 98

A program to generate a reference sine wave corresponding to blade rate and shaft rate from a tachometer pulse. General I/O scheme based on TT.c, by Analog Devices.

/* **** */

// includes:

// ADSP-2106x System Register bit definitions

#include <def21060.h>

#include <21060.h>

#include <signal.h>

#include <sport.h>

#include <macros.h>

// Included user files

// "Sine6.h" is a listing of 6125 values of a sine wave, sampled at even intervals. The

// text of the file is not included here.

#include "Sine6.h"

// The text of "TTRefSig.h" is included below

#include "TTRefSig.h"

/* **** */

TTRefSig.h

Header file for TTRefSig.c

/* **** */

// defines:

```

// Processor I/O definitions
#define CP_PCI 0x20000 // Program-Controlled Interrupts bit
#define CP_MAF 0x1fff // Valid memory address field bits

#define SetIOP( addr, val ) ( *( int * )addr ) = ( val )
#define GetIOP( addr ) ( *( int * )addr )

#define NUM_1847Regs 16 // Number of control registers in the Codec

// Minimum input value that will be considered a pulse from the tachometer
#define THRESHOLD 265 // Threshold value for pulse detection

// Output gain
#define GAIN 256 // Amplitude of output sine waves

// Number of blades on the fan (for calculation of blade rate)
#define BLADES 3 // Number of blades on propeller

```

```

/*****

```

```

// structs:

```

```

// DMA chaining Transfer Control Blocks (Processor I/O)

```

```

typedef struct

```

```

{
    unsigned lpath3; // for mesh multiprocessing
    unsigned lpath2; // for mesh multiprocessing
    unsigned lpath1; // for mesh multiprocessing
    unsigned db; // General purpose register
    unsigned gp; // General purpose register
    unsigned **cp; // Chain Pointer to next TCB
    unsigned c; // Count register
    int im; // Index modifier register
    unsigned *ii; // Index register
} _tcb;

```

```

/*****

```

```

Global variables.

```

Some of these have to be declared as "volatile" so that the compiler doesn't optimize them away when they appear to be unused.

The I/O functions of the DSP are handled through a device known as a Codec (Code/Decode). The following variables define the behavior of the Codec.

regs1847 contains the values that each of the sixteen 1847 control registers will be set to. These sixteen bit words will be sent to the Codec in order during initialization on channel zero (the control channel), and will thus go into the one sixteen bit control word input register. The first four bits of this register are set as 1100 (0xc) in all but the last word to be transmitted so that the Mode Change Enable bit will be set. When register fifteen is transmitted, the MCE bit is cleared, so that certain properties of the Codec cannot be modified. The second four bits are used for indirect addressing, so that features of each of the thirteen eight bit control registers can be set (not that registers 11, 14, and 15 do not exist). The final eight bits will be loaded into the appropriate eight bit control register.

Register	Effect
0	determines which source is connected to the left input, and the amplification on that source
1	same as 0, but for right input
2	left aux 1 input control
3	right aux 1 input control
4	left aux 2 input control
5	right aux 2 input control
6	Left output control--sets attenuation/muting
7	Right output control--sets attenuation/muting
8	Data format--sets compression and signed/unsigned format
9	Interface configuration register--controls autocalibration and playback
10	Pin control register--control clock/crystal output
11	not a register (writing here has no effect)
12	Miscellaneous--sets which TDM slots the 1847 transmits on, and the frame size
13	Digital Mix Control--controls digital mixing of output and input
14	not a register (writing here has no effect)
15	not a register (writing here has no effect)

*****/

```

volatile int    txCount,
               *txPtr,
               samples    =    0,
               pulseOn    =    0;

int    cmd_blk[8],           // command block
       rxBuf[3],             // receive buffer (three channels)
       txBuf[3]    = {      0xcc40, // transmit buffer (three channels)
                          0,
                          0 },
       regs1847[NUM_1847Regs] = { 0xc008, // index 0 - left input control
                                   0xc108, // index 1 - right input control
                                   0xc280, // index 2 - left aux 1 input
                                       // control
                                   0xc380, // index 3 - right aux 1 input
                                       // control
                                   0xc480, // index 4 - left aux 2 input
                                       // control
                                   0xc580, // index 5 - right aux 2 input
                                       // control
                                   0xc600, // index 6 - left dac control
                                   0xc700, // index 7 - right dac control
                                   0xc84f, // index 8 - data format
                                   0xc909, // index 9 - interface
                                       // configuration
                                   0xca00, // index 10 - pin control
                                   0xcb00, // index 11 - no register
                                   0xcc40, // index 12 - miscellaneous
                                       // information
                                   0xcd00, // index 13 - digital mix
                                       control
                                   0xce00, // index 14 - no register
                                   0xf000 }; // index 15 - no register

volatile float    deltaS    =    0,
                  deltaB    =    0;

_tcb              rx_tcb    = {    0, 0, 0, 0, 0, 0, 3, 1, 0 }, // receive tcb
                  tx_tcb    = {    0, 0, 0, 0, 0, 0, 3, 1, 0 }; // transmit tcb

```



```

/*****/

// function prototypes:

void Init21k( void );
void Spt0Asserted( int sigNum );
void Spr0Asserted( int sigNum );
void SetupSports ( void );
void Config1847( void );

//      END OF TTREFSIG.H

//      RESUME TTREFSIG.C

/*****
*****  main() *****
*****
*      The program. The main loop processes the input values to determine when
*      a pulse has occurred, and to count the time between pulses. It can then
*      adjust the delta values in computing the next sine wave sample that needs
*      to be output.
*****/

void main ( void )
{
    int    finish,
          x;

    // Initialize some SHARC registers.
    Init21k();

    // Reset the Codec.
    set_flag( SET_FLAG0, CLR_FLAG );      // Put CODEC into RESET
    for( x=0 ; x<0xffff ; x++ );          // Hold CODEC in RESET for a while
    set_flag( SET_FLAG0, SET_FLAG );      // Release CODEC from RESET

    // Configure SHARC serial port.
    SetupSports();

    // Send setup commands to CODEC.

```

```

Config1847();

// Loop forever.
while( 1 )
{
    // the variable "pulseOn" is set by the SPORT receive interrupt handler
    while ( !pulseOn );           // Wait for a pulse to arrive
    while( pulseOn );             // Wait for the end of the pulse

    // the variable "samples" is incremented by the receive interrupt handler
    // each time it receives an interrupt. By counting the number of samples
    // that occur in a full pulse cycle, the frequency of the tachometer output
    // (or the shaft rate) can be determined.
    finish = samples;             // number of samples after one complete cycle
    samples = 0;                  // Reset the count

    // These increments are used to jump through the table of sine wave
    // sample values included in "Sine6.h". By changing the size of the
    // increments, the number of samples required to pass through a full
    // cycle of a sine wave can be changed. The frequency of the DSP
    // output sine wave is thus affected by the size of deltaS and deltaB
    deltaS = TABLE_Length / finish; // Shaft rate increment
    deltaB = BLADES * deltaS;        // Blade rate increment
}
}

/*****
***** Init21k() *****
*****
*   Sets the control channel transmit information to transmit the desired
*   state of all of the control registers (as stored in the array regs1847),
*   but does not yet call for the information to be transmitted (txCount = 0),
*   since the SPORTs haven't been set up yet. Turns on interrupt nesting and turns
*   off the reset bit for the Codec.
*****/

void Init21k( void )
{
    // Initialize pointer and counter to transmit commands.
    txCount = 0;

```

```

    txPtr = regs1847;

    // Enable interrupt nesting.
    asm( "#include <def21060.h>" );
    asm( "bit set mode1 NESTM;" );

    // Turn flag LEDs off.
    set_flag( SET_FLAG2, SET_FLAG );

    return;
}

/*****
*****      Spt0Asserted()
*****
*   SPORT0 transmit interrupt handler--Serial port transmit complete.
*
*   This function has three missions, corresponding to the three time
*   multiplexed SPORT transmit channels. The control channel (txBuf[0]) will
*   only be used at the very beginning of the program, to transmit
*   the desired control register contents to the Codec.
*
*   Channels one and two carry the shaft- and blade-rate sinusoids
*   respectively. Both sine waves are calculated by stepping through a common
*   lookup table stored in "table" and defined in the included "Sine6.h" file.
*   The increments (deltaS and deltaB) are calculated in main() and stored in
*   global floating point variables. The instantaneous value of each sine
*   wave is obtained by dereferencing a pointer to some location in "table", as
*   determined by adding the appropriate increment to the previous value of the
*   pointer. This addition is stored in a floating point variable to prevent
*   frequency errors which would be introduced by ignoring the fractional
*   portion of the increment value. This floating point number is then cast
*   to a pointer, which can be dereferenced to retrieve a sine wave sample
*   value.
*****/

void Spt0Asserted( int sigNum )
{
    static int    initialized    = 0;
    static float  floatSAddress = 0,

```

```

        floatBAddress = 0,
        *sAddress      = table,
        *bAddress      = table;

// Set starting addresses for floating point accuracy pointers
if ( !initialized )
{
    floatSAddress = ( float )( int )sAddress;
    floatBAddress = ( float )( int )bAddress;
    initialized = 1;
}

// Check if there are more commands left to transmit.
if( txCount )
{
    // If so, put the comand into the transmit buffer and update count.
    txBuf[0] = *txPtr++;
    txCount--;
}

// Transmit shaft rate on channel one
floatSAddress += deltaS;
if ( floatSAddress >= table + TABLE_Length )
    floatSAddress -= TABLE_Length; // Wrap around when we pass the
                                    // end of the table

sAddress = ( *float )( int )floatSAddress;
txBuf[1] = ( int )( GAIN * ( *sAddress ) );

// Transmit blade rate on channel two
floatBAddress += deltaB;
if ( floatBAddress >= table + TABLE_Length )
    floatBAddress -= TABLE_Length; // Wrap around when we pass the
                                    // end of the table

bAddress = ( *float )( int )floatBAddress;
txBuf[2] = ( int )( GAIN * ( *bAddress ) );
}

```

```

/*****
***** Spr0Asserted() *****/
*****
*   SPORT0 receive interrupt handler.
*
*   Tests the input signal to determine if we're in the midst of a tachometer
*   pulse. The value of "pulseOn" is used in main() to determine the tach
*   frequency. "samples" is used to count how many samples occur between
*   pulses (it is reset to zero in main() when appropriate).
*****/

```

```

void Spr0Asserted( int sigNum )
{
    if ( rxBuf[1] < THRESHOLD )
        pulseOn = 1;
    else
        pulseOn = 0;

    samples++;
}

```

```

/*****
***** SetupSports() *****/
*****
*   Configures the DSP serial ports for I/O
*****/

```

```

void SetupSports ( void )
{
    // Configure SHARC serial port SPORT0

    // Multichannel communications setup
    sport0_iop.mtcs    = 0x00070007;    // transmit on words 0,1,2,16,17,18
    sport0_iop.mrcs    = 0x00070007;    // receive on words 0,1,2,16,17,18
    sport0_iop.mtccs   = 0x00000000;    // no companding on transmit
    sport0_iop.mrccs   = 0x00000000;    // no companding on receive

    // TRANSMIT CONTROL REGISTER
    // STCTL0 <= 0x001c00f2

```

```

sport0_iop.txc.mdf      = 1; // multichannel frame delay (MFD)
sport0_iop.txc.schen    = 1; // Tx DMA chaining enable
sport0_iop.txc.sden     = 1; // Tx DMA enable
sport0_iop.txc.lafs     = 0; // Late TFS (alternate)
sport0_iop.txc.ltfs     = 0; // Active low TFS
sport0_iop.txc.ditfs    = 0; // Data independent TFS
sport0_iop.txc.itfs     = 0; // Internally generated TFS
sport0_iop.txc.tfsr     = 0; // TFS Required

sport0_iop.txc.ckre     = 0; // Data and FS on clock rising edge
sport0_iop.txc.gclk     = 0; // Enable clock only during transmission
sport0_iop.txc.iclk     = 0; // Internally generated Tx clock
sport0_iop.txc.pack     = 0; // Unpack 32b words into two 16b tx's

sport0_iop.txc.slen = 15; // Data word length minus one
sport0_iop.txc.sendn = 0; // Data word endian 1 = LSB first
sport0_iop.txc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
sport0_iop.txc.spen = 0; // Enable (clear for MC operation)

```

// RECEIVE CONTROL REGISTER

// SRCTL0 <= 0x1f8c20f2

```

sport0_iop.rxc.nch      = 31; // multichannel number of channels - 1
sport0_iop.rxc.mce      = 1; // multichannel enable
sport0_iop.rxc.spl      = 0; // Loop back configure (test)
sport0_iop.rxc.d2dma    = 0; // Enable 2-dimensional DMA array
sport0_iop.rxc.schen    = 1; // Rx DMA chaining enable
sport0_iop.rxc.sden     = 1; // Rx DMA enable
sport0_iop.rxc.lafs     = 0; // Late RFS (alternate)
sport0_iop.rxc.ltfs     = 0; // Active low RFS
sport0_iop.rxc.irfs     = 0; // Internally generated RFS
sport0_iop.rxc.rfsr     = 1; // RFS Required
sport0_iop.rxc.ckre     = 0; // Data and FS on clock rising edge
sport0_iop.rxc.gclk     = 0; // Enable clock only during transmission
sport0_iop.rxc.iclk     = 0; // Internally generated Rx clock
sport0_iop.rxc.pack     = 0; // Pack two 16b rx's into 32b word

sport0_iop.rxc.slen     = 15; // Data word length minus one
sport0_iop.rxc.sendn = 0; // Data word endian 1 = LSB first
sport0_iop.rxc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
sport0_iop.rxc.spen     = 0; // Enable (clear for MC operation)

```

```

// Enable sport0 xmit & rcv irqs (DMA enabled)
interrupt(SIG_SPR0I, Spr0Asserted);
interrupt(SIG_SPT0I, Spt0Asserted);

// Set up Transmit Transfer Control Block for chained DMA
tx_tcb.ii = txBuf;           // DMA source buffer address
tx_tcb.cp = &tx_tcb.ii;      // define ptr to next TCB (point to self)
SetIOP(CP2, (((int)&tx_tcb.ii) & CP_MAF) | CP_PCI);
                        // define ptr to current TCB (kick off DMA)
                        // (SPORT0 transmit uses DMA ch 2)

// Set up Receive Transfer Control Block for chained DMA
rx_tcb.ii = rxBuf;           // DMA destination buffer address
rx_tcb.cp = &rx_tcb.ii;      // define ptr to next TCB (point to self)
SetIOP(CP0, (((int)&rx_tcb.ii) & CP_MAF) | CP_PCI);
                        // define ptr to current TCB (kick off DMA)
                        // (SPORT0 receive uses DMA ch 0)
}

/*****
***** Config1847() *****
*****
*   Sets up the SPORT transmit buffer to send control register values to the
*   Codec.  Waits for all control values to be sent, then waits for the Codec
*   to finish its autocalibration (determined by examining the status word,
*   which is sent by the Codec over channel zero).
*****/

void Config1847( void )
{
    // Set up pointer and counter to transmit commands.
    txPtr = regs1847;
    txCount = NUM_1847Regs;

    // Wait for all commands to be transmitted.
    while( txCount )
        idle();

    // Wait for AD1847 autocal to start.
    while( !(rxBuf[0] & 0x0002) )

```

```
        idle();

        // Wait for AD1847 autocal to finish.
        while( rxBuf[0] & 0x0002 )
            idle();

        return;
    }
```


APPENDIX I—FILTERED-X PROGRAM

```

/*****

slp.c

by John Wiggins      20 March 98

An implementation of the filtered-x LMS algorithm for the SHARC EZ-Kit DSP Board

*****/

// includes:

#include <def21060.h>
#include <21060.h>
#include <signal.h>
#include <sport.h>
#include <macros.h>

// The text of "slp.h" is included here
#include "slp.h"

/*****

slp.h

Header file for slp.c

*****/

// defines:

// DMA Chain pointer bit definitions (processor I/O)
#define CP_PCI      0x20000    // Program-Controlled Interrupts bit
#define CP_MAF      0x1fff     // Valid memory address field bits

#define SetIOP( addr, val )    ( *( int * )addr ) = ( val )
#define GetIOP( addr )        ( *( int * )addr )

#define NUM_1847Regs    16    // Number of control registers in the Codec

```

```
#define      MU_          0.00001
#define      NUM_Taps     100
```

```
/**
*****
**/
```

```
// structs:
```

```
// DMA chaining Transfer Control Blocks
```

```
typedef struct
```

```
{
    unsigned    lpath3;    // for mesh multiprocessing
    unsigned    lpath2;    // for mesh multiprocessing
    unsigned    lpath1;    // for mesh multiprocessing
    unsigned    db;        // General purpose register
    unsigned    gp;        // General purpose register
    unsigned    **cp;      // Chain Pointer to next TCB
    unsigned    c;        // Count register
    int         im;        // Index modifier register
    unsigned    *ii;       // Index register
} _tcb;
```

```
/**
*****
**/
```

Global variables.

Some of these have to be declared as "volatile" so that the compiler doesn't optimize them away when they appear to be unused.

The I/O functions of the DSP are handled through a device known as a Codec (Code/Decode). The following variables define the behavior of the Codec.

regs1847 contains the values that each of the sixteen 1847 control registers will be set to. These sixteen bit words will be sent to the Codec in order during initialization on channel zero (the control channel), and will thus go into the one sixteen bit control word input register. The first four bits of this register are set as 1100 (0xc) in all but the last word to be transmitted so that the Mode Change Enable bit will be set.

When register fifteen is transmitted, the MCE bit is cleared, so that certain properties of the Codec cannot be modified. The second four bits are used for indirect addressing, so that features of each of the thirteen eight bit control registers can be set (not that registers 11, 14, and 15 do not exist). The final eight bits will be loaded into the appropriate eight bit control register.

Register	Effect
0	determines which source is connected to the left input, and the amplification on that source
1	same as 0, but for right input
2	left aux 1 input control
3	right aux 1 input control
4	left aux 2 input control
5	right aux 2 input control
6	Left output control--sets attenuation/muting
7	Right output control--sets attenuation/muting
8	Data format--sets compression and signed/unsigned format
9	Interface configuration register--controls autocalibration and playback
10	Pin control register--control clock/crystal output
11	not a register (writing here has no effect)
12	Miscellaneous--sets the TDM slots the 1847 transmits on, and the frame size
13	Digital Mix Control--controls digital mixing of output and input
14	not a register (writing here has no effect)
15	not a register (writing here has no effect)

*****/

```

int  cmd_blk[8],           // command block
      rxBuf[3],            // receive buffer (three channels)
      txBuf[3] = { 0xcc40, // transmit buffer (three channels)
                   0,
                   0 },
      regs1847[NUM_1847Regs] = { 0xc008, // index 0 - left input control
                                  0xc108, // index 1 - right input control
                                  0xc280, // index 2 - left aux 1 input
                                       // control
                                  0xc380, // index 3 - right aux 1 input
                                       // control
                                  0xc480, // index 4 - left aux 2 input
                                       // control

```

```

0xc580,      // index 5 - right aux 2 input
              // control
0xc600,      // index 6 - left dac control
0xc700,      // index 7 - right dac control
0xc84f,      // index 8 - data format
0xc909,      // index 9 - interface
              // configuration
0xca00,      // index 10 - pin control
0xcb00,      // index 11 - no register
0xcc40,      // index 12 - miscellaneous
0xcd00,      // index 13 - digital mix
              // control
0xce00,      // index 14 - no register
0xf000 };    // index 15 - no register

```

```

volatile int  tx0Count,
              *tx0Ptr;

```

```

#define      FILT_NB      1      // These values must be less than NUM_Taps
#define      FILT_NA      1      // Vectors must be of the same length for the
                                  // filtering algorithm to work properly

```

```

// The filtered-x secondary path model numerator and denominator (differing values
// were used for experimentation)

```

```

const float  xFiltB[FILT_NB]      = { 1 },
              xFiltA[FILT_NA]      = { 1 };

```

```

volatile float e,
              tap[NUM_Taps]        = {};

```

```

float        x[NUM_Taps]           = {},
              r[NUM_Taps]           = {},
              watchX[NUM_Taps] = {},
              watchR[NUM_Taps] = {};

```

```

_tcb         rx_tcb                = { 0, 0, 0, 0, 0, 0, 3, 1, 0 },    // receive tcb
              tx_tcb                = { 0, 0, 0, 0, 0, 0, 3, 1, 0 };    // transmit tcb

```

```

/*****

```

```
// Set up circular buffers for x[] and r[]
CIRCULAR_BUFFER( float, 5, xPtr );
CIRCULAR_BUFFER( float, 2, rPtr );
```

```
// function prototypes:
void Init21k( void );
void Spt0Asserted( int sigNum );
void Spr0Asserted( int sigNum );
void SetupSports ( void );
void Config1847( void );
```

```
//      END SLP.H
```

```
//      RESUME SLP.C
```

```

/*****
*****                               main() *****
*****
*****
*      After initializing various DSP components, the program steps into an endless
*      loop, where it performs the signal processing job in the active noise control
*      system. The filtering consists of five main steps:
*
*          1) Reading the input data (reference and error signals)
*          2) Filtering the input data according to the filtered-x algorithm
*          3) Running the FIR filter to generate a control value for the bearings
*          4) Transmitting the control value to the bearing controller
*          5) Updating the FIR filter according to the LMS algorithm
*
*      In order to speed the execution of the program, a buffering scheme called
*      "circular buffering" is used to store input values.
*****/
```

```
int    go = 0;
float  in1,
       in2;
```

```
void main ( void )
{
    int    x,
           index;
```

```

float  e,
      xVal,
      rVal,
      newR,
      out1,
      out2;

// Initialize some SHARC registers.
Init21k();

// Reset the Codec.
set_flag( SET_FLAG0, CLR_FLAG );      // Put CODEC into RESET
for( x = 0; x < 0xffff; x++ );        // Hold CODEC in RESET for a while
set_flag( SET_FLAG0, SET_FLAG );      // Release CODEC from RESET

// Configure SHARC serial port.
SetupSports();

// Send setup commands to CODEC.
Config1847();

// Loop forever.
while ( 1 )
{
    while ( !go );      // Wait for a Rx interrupt (handler will set go = 1)
    go = 0;

    // Since old reference signal inputs will be used again, the reference
    // input values are stored in a buffer
    CIRC_MODIFY( xPtr, 1 ); // Point circular buffer to oldest x value
    CIRC_WRITE( xPtr, 0, in1, dm ); // Write the current reference input
                                   // over it

    // Only the current error value is used, so it is not buffered
    e = in2;                      // Get the current error signal

    // Filter the input signal according to the filtered-x algorithm
    //  $r = (X * xFiltB - R * xFiltA) / xFiltA(0)$ 
    newR = 0;
    for ( index = 0; index < FILT_NB; index++ )

```

```

{
    CIRC_READ( xPtr, -1, xVal, dm );
    newR += xFiltB[index] * xVal;
}

// Return the circular buffer pointer to the current x value
CIRC_MODIFY( xPtr, FILT_NB );

// The circular buffer for r[] is already pointing to the most recent value, as
// we haven't modified it since writing the last r value
for ( index = 1; index < FILT_NA; index++ )
{
    CIRC_READ( rPtr, -1, rVal, dm );
    newR -= xFiltA[index] * rVal;
}

/* Now we advance the r[] pointer one more place than we backed it up
during the above for loop, so that it points to the oldest r value. Since we
started the above loop at 1, not 0, this still means advancing it by the
number of A coefficients */
CIRC_MODIFY( rPtr, FILT_NA );

// And write the newest r value over the oldest
CIRC_WRITE( rPtr, 0, newR / xFiltA[0], dm );

// Run the adaptive filter:
//      output = TAP * X,       $W(n+1) = W(n) + \mu * R * e$ 

out1 = 0;      // Blade rate control channel
out2 = 0;      // Unused control channel
for ( index = 0; index < NUM_Taps; index++ )
{
    // determine the output signal by filtering the reference input:
    CIRC_READ( xPtr, -1, xVal, dm );
    out1 += tap[index] * xVal;

    // and update each weight with the LMS algorithm after its use
    CIRC_READ( rPtr, -1, rVal, dm );
    tap[index] += MU_ * rVal * e;
}
// The circular buffer pointers will have made a complete loop and again

```

```

        // be pointing to the current x and r values at the completion of this for
        // loop.

        // Transmit the filter output
        txBuf[1] = out1;
        txBuf[2] = out2;
    }
}

/*****
*****      Init21k()      *****
*****
*       Sets the control channel transmit information to transmit the desired
*       state of all of the control registers (as stored in the array regs1847),
*       but does not yet call for the information to be transmitted (tx0Count = 0),
*       since the SPORTs haven't been set up yet. Turns on interrupt
*       nesting and turns off the reset bit for the Codec.
*****/

void Init21k( void )
{
    // Initialize pointer and counter to transmit commands.
    tx0Count = 0;
    tx0Ptr = regs1847;

    // Enable interrupt nesting.
    asm( "#include <def21060.h>" );
    asm( "bit set model NESTM;" );

    // Turn flag LEDs off.
    set_flag( SET_FLAG2, SET_FLAG );

    return;
}

/*****
*****      Spt0Asserted() *****
*****
*       SPORT0 transmit interrupt handler--Serial port transmit complete.
*****/

```



```

*
*   This function is used to transmit the desired control register contents to the
*   Codec.
*
*   Transmission of control data to the magnetic bearings is handled in main()
*****/

```

```

void Spt0Asserted( int sigNum )
{
    // Check if there are more commands left to transmit.
    if( tx0Count )
    {
        // If so, put the comand into the transmit buffer and update count.
        txBuf[0] = *tx0Ptr++;
        tx0Count--;
    }
}

```

```

/*****
***** Spr0Asserted() *****
*****
*   SPORT0 receive interrupt handler.
*
*   This interrupt occurs at even intervals, based on the sample time of the Codec.
*   The signal processing loop in main() waits for a receive interrupt to occur before
*   beginning each iteration, so that the timing of the processing remains constant.
*   This interrupt handler sets a flag ("go") which is checked in main() to determine
*   when a receive interrupt has occurred.
*****/

```

```

void Spr0Asserted( int sigNum )
{
    static int    cnt    = 0;

    float         in1    = 0,
                 in2    = 0;

    in1 = rxBuf[1];
    in2 = rxBuf[2];

```

```

    go = 1;
}

```

```

/*****
*****      SetupSports() *****
*      Configure the DSP serial ports for I/O.
*****/

```

```

void SetupSports ( void )
{

```

```

    // Configure SHARC serial port SPORT0

```

```

    // Multichannel communications setup

```

```

    sport0_iop.mtcs      = 0x00070007;    // transmit on words 0,1,2,16,17,18
    sport0_iop.mrcs      = 0x00070007;    // receive on words 0,1,2,16,17,18
    sport0_iop.mtccs     = 0x00000000;    // no companding on transmit
    sport0_iop.mrccs     = 0x00000000;    // no companding on receive

```

```

    // TRANSMIT CONTROL REGISTER

```

```

    // STCTL0 <= 0x001c00f2

```

```

    sport0_iop.txc.mdf      = 1;    // multichannel frame delay (MFD)
    sport0_iop.txc.schen    = 1;    // Tx DMA chaining enable
    sport0_iop.txc.sden     = 1;    // Tx DMA enable
    sport0_iop.txc.lafs     = 0;    // Late TFS (alternate)
    sport0_iop.txc.ltfs     = 0;    // Active low TFS
    sport0_iop.txc.ditfs    = 0;    // Data independent TFS
    sport0_iop.txc.itfs     = 0;    // Internally generated TFS
    sport0_iop.txc.tfsr     = 0;    // TFS Required

    sport0_iop.txc.ckre     = 0;    // Data and FS on clock rising edge
    sport0_iop.txc.gclk     = 0;    // Enable clock only during transmission
    sport0_iop.txc.iclk     = 0;    // Internally generated Tx clock
    sport0_iop.txc.pack     = 0;    // Unpack 32b words into two 16b tx's

    sport0_iop.txc.slen     = 15;    // Data word length minus one
    sport0_iop.txc.sendn    = 0;    // Data word endian 1 = LSB first
    sport0_iop.txc.dtype    = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
    sport0_iop.txc.spen     = 0;    // Enable (clear for MC operation)

```

// RECEIVE CONTROL REGISTER

// SRCTL0 <= 0x1f8c20f2

```

sport0_iop.rxc.nch      = 31; // multichannel number of channels - 1
sport0_iop.rxc.mce      = 1;  // multichannel enable
sport0_iop.rxc.spl      = 0;  // Loop back configure (test)
sport0_iop.rxc.d2dma    = 0;  // Enable 2-dimensional DMA array
sport0_iop.rxc.schen    = 1;  // Rx DMA chaining enable
sport0_iop.rxc.sden     = 1;  // Rx DMA enable
sport0_iop.rxc.lafs     = 0;  // Late RFS (alternate)
sport0_iop.rxc.ltfs     = 0;  // Active low RFS
sport0_iop.rxc.irfs     = 0;  // Internally generated RFS
sport0_iop.rxc.rfsr     = 1;  // RFS Required
sport0_iop.rxc.ckre     = 0;  // Data and FS on clock rising edge
sport0_iop.rxc.gclk     = 0;  // Enable clock only during transmission
sport0_iop.rxc.iclk     = 0;  // Internally generated Rx clock
sport0_iop.rxc.pack     = 0;  // Pack two 16b rx's into 32b word

sport0_iop.rxc.slen     = 15; // Data word length minus one
sport0_iop.rxc.sendn    = 0;  // Data word endian 1 = LSB first
sport0_iop.rxc.dtype    = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
sport0_iop.rxc.spen     = 0;  // Enable (clear for MC operation)

```

```

// Set up the circular buffers for the reference and filtered-reference inputs
// before enabling interrupts

```

BASE(xPtr) = x;

BASE(rPtr) = r;

LENGTH(xPtr) = NUM_Taps;

LENGTH(rPtr) = NUM_Taps;

// Enable sport0 xmit & rcv irqs (DMA enabled)

interrupt(SIG_SPR0I, Spr0Asserted);

interrupt(SIG_SPT0I, Spt0Asserted);

// Set up Transmit Transfer Control Block for chained DMA

tx_tcb.ii = txBuf; // DMA source buffer address

tx_tcb.cp = &tx_tcb.ii; // define ptr to next TCB (point to self)

SetIOP(CP2, (((int)&tx_tcb.ii) & CP_MAF) | CP_PCI);

// define ptr to current TCB (kick off DMA)

// (SPORT0 transmit uses DMA ch 2)

```

// Set up Receive Transfer Control Block for chained DMA
rx_tcb.ii = rxBuf;           // DMA destination buffer address
rx_tcb.cp = &rx_tcb.ii;     // define ptr to next TCB (point to self)
SetIOP(CP0, (((int)&rx_tcb.ii) & CP_MAF) | CP_PCI);
                           // define ptr to current TCB (kick off DMA)
                           // (SPORT0 receive uses DMA ch 0)
}

/*****
***** Config1847() *****/
*****
*   Sets up the SPORT transmit buffer to send control register values to the
*   Codec.  Waits for all control values to be sent, then waits for the Codec
*   to finish its autocalibration (determined by examining the status word,
*   which is sent by the Codec over channel zero).
*****/

void Config1847( void )
{
    // Set up pointer and counter to transmit commands.
    tx0Ptr = regs1847;
    tx0Count = NUM_1847Regs;

    // Wait for all commands to be transmitted.
    while( tx0Count )
        idle();

    // Wait for AD1847 autocal to start.
    while( !( rxBuf[0] & 0x0002 ) )
        idle();

    // Wait for AD1847 autocal to finish.
    while( rxBuf[0] & 0x0002 )
        idle();

    return;
}

```

REFERENCES

- [1] Allaire, P. E., et al., "Magnetic Thrust Bearing Operation and Industrial Pump Application", Journal of Engineering for Gas Turbines and Power, vol. 119, Jan. 97, pp. 168-73.
- [2] Bendat, J. S., and Piersol, A. G., Engineering Applications of Correlation and Spectral Analysis, ©1993 John Wiley & Sons, Inc., New York.
- [3] Dorf, R. C., and Bishop, R. H., Modern Control Systems, ©1995 Addison-Wesley Publishing Co., Reading, Massachusetts.
- [4] Elliott, S. J., et al., "The Behavior of a Multiple Channel Active Control System", IEEE Transactions on Signal Processing, v. 40, May 1992, pp. 1041-52.
- [5] Elliott, S. J., and Nelson, P. A., "Active Noise Control", IEEE Signal Processing Magazine, October 93, pp. 12-35.
- [6] Eriksson, L. J., "A Primer on Active Sound and Vibration Control", Sensors, February 97, pp. 18-31.
- [7] Eriksson, L. J., "Development of the Filtered-U Algorithm for Active Noise Control", The Journal of the Acoustical Society of America, v. 89, January 1991, pp. 257-65.
- [8] Friedland, B., Control System Design, ©1986 McGraw-Hill, Inc., New York.
- [9] Gosling, R. J., Oliva, A. F., & Church, K. B. (1988). The AMPS nuclear reactor based air-independent power source for diesel electric submarines. Warship '88 International Symposium on Conventional Naval Submarines, London, 2.
- [10] Jenkins, K. W., et al., Advanced Concepts in Adaptive Signal Processing, ©1996, Kluwer Academic Publishers, Massachusetts.
- [11] Kousen, K. A., and Verdon, J. M., "Active Control of Wake/Blade-Row Interaction Noise", AIAA Journal, v. 32, October 1994, pp. 1953-60.
- [12] Kuo, S. M., and Morgan, D. R., Active Noise Control Systems, ©1996 John Wiley & Sons, Inc., New York.
- [13] Kuo, B. C., Digital Control Systems, ©1992, Saunders College Publishing, Ft. Worth.

- [14] Lauchle, G. C., et al., "Active Control of Axial-Flow fan Noise", The Journal of the Acoustical Society of America, v. 101, January 1997, pp. 341-49.
- [15] Lester, H. C., and Fuller, C. R., "Active Control of Propeller-Induced Noise Fields Inside a Flexible Cylinder", AIAA Journal, v. 28, August 1990, pp. 1374-80.
- [16] Minter, J. A., and Fleeter, S., "Active Control of Turbomachinery Discrete Frequency Noise Utilizing Oscillating Flaps and Pistons", AIAA Journal, v. 35, July 1997, pp. 1105-12.
- [17] Nelson, P. A., and Elliot, S. J., Active Control of Sound, ©1992 Harcourt Brace & Company, London.
- [18] Piper, G. E., and Calvert, T. E., "Active Fluidborne Noise Control of a Magnetic Bearing Pump", Proceedings of the ASME Noise Control and Acoustics Division, Vol. 21, 1995, pp. 55-76.
- [19] Regalia, P. A., "Numerical Stability Issues in Fast Least-Squares Adaptation Algorithms", Optical Engineering, v. 31, June 1992, PP. 1144-52.
- [20] Risi, J. D., et al., "Analytical Investigation of Active Control of Radiated Inlet Fan Noise", The Journal of the Acoustical Society of America, v. 99, January 1996, pp. 408-16.
- [21] Rotrubin, J. L., Cousquer, J., Khoury, M., & Curtoud, P. M. (1991). Hydro-acoustic analysis of external structure of submarine. Warship '91 International Symposium on Naval Submarines, London, 1.
- [22] Schweitzer, G., et al., Active Magnetic Bearings, ©1994 vdf, Hochschulverlag.
- [23] Shiau, T. N., et al., "Vibration and Control of a Flexible Rotor in Magnetic Bearings Using Hybrid Method and H^∞ Control Theory", Journal of Engineering for Gas Turbines and Power, vol. 119, Jan. 97, pp. 178-85.
- [24] Sievers, L. A., and von Flotow, A. H., "Comparison and Extensions of Control Methods for Narrow-Band Disturbance Rejection", IEEE Transactions on Signal Processing, v. 40, October 1992, pp. 2377-91.
- [25] Simonich, J., et al., "Active Aerodynamic Control of Wake-Airfoil Interaction Noise—Experiment", AIAA Journal, v. 31, October 1993, pp. 1761-68.
- [26] Sullivan, M., "Adaptive Filters Handle Changing Signals for Noise Removal, Signal Prediction", Personal Engineering, December 1995, pp. 61-63.

- [27] Vaseghi, Saeed V., Advanced Signal Processing and Digital Noise Reduction, ©1996, John Wiley and Sons, West Sussex, England, and B. G. Teubner, Stuttgart, Germany.
- [28] Widrow, B., et al., Adaptive Signal Processing, ©1985, Prentice Hall.